УДК 519.686

# АВТОМАТИЗАЦИЯ ОТЛАДКИ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

**В. А. Крюков**[1], **М. В. Кудрявцев**[2]

Статья посвящена новым результатам работ, направленных на создание автоматизированных средств отладки параллельных программ в моделях с глобальным адресным пространством. Работа поддержана грантом Президента РФ для ведущих научных школ (код проекта НШ–383.2006.9) и грантами РФФИ (коды проектов 05–01–00678 и 05–07–90026).

**Введение.** С распространением многопроцессорных, многоядерных и кластерных технологий разработка параллельных программ становится все более актуальной. Развитые инструменты отладки способны существенно облегчить и ускорить их разработку.

Ручные методы отладки, основанные на точках останова и отладочных печатях, неадекватны сложности многих реальных научно-технических приложений. Автоматизированные методы — сравнительная отладка и динамический анализ корректности — могли бы существенно облегчить и ускорить отладку этих приложений. Динамический анализ эффективен для поиска неверных спецификаций параллелизма, реальных и потенциальных дедлоков, некорректного доступа к общим данным, ошибочных обращений и последовательностей обращений к библиотекам, выходов за границы массивов и прочих ошибок [2 – 10]. Сравнительная отладка заключается в сравнении двух запусков программы — эталонного и отлаживаемого. Обнаруженные при этом различия используются для локализации ошибок [9 – 15]. Оба этих метода отладки параллельных программ были предложены в середине 90-х годов XX столетия, реализованы в DVM-системе [1] и показали высокую эффективность при отладке приложений с ограниченным объемом данных и вычислений.

Для автоматизации отладки реальных научно-технических приложений можно было бы воспользоваться методами сравнительной отладки и анализа корректности, если бы удалось найти способы существенного сокращения (в сотни и тысячи раз) необходимых для них ресурсов памяти и времени. В первом и втором разделах статьи описаны предложенные нами методы такого сокращения ресурсов, приведены результаты экспериментов. В третьем разделе рассмотрен метод, позволяющий применять сравнительную отладку к параллельным приложениям, недетерминизм которых вызван изменением порядка выполнения операций при редукционном суммировании и умножении.

**1. Метод граничных итераций и двойных тел циклов.** Этот метод направлен на существенное сокращение ресурсов, требуемых для сравнительной отладки и динамического анализа корректности. В его основу положены две простые идеи.

Первая — сократить затраты на выполнение контроля при многократном прохождении одной и той же точки программы. Каждый оператор программы (а следовательно, и соответствующие точки контроля используемых и модифицируемых в этом операторе переменных) может входить в тело нескольких охватывающих его циклов и выполняться на каждой итерации такого многомерного цикла. Предлагается проводить контроль не на всех итерациях, а только на граничных.

Вторая идея — чтобы избавиться от накладных расходов, вызванных проверками необходимости контроля каждого выполняющегося оператора программы, предлагается для каждого цикла генерировать два варианта его тела: с отладочной инструментацией и без нее.

**1.1. Описание метода.** Большинство научно-технических приложений состоит из циклов обработки массивов. При обработке массивов наиболее вероятным местом возникновения ошибок (например, использования неинициализированных переменных, выходов за границы массива и др.) являются граничные итерации. Кроме того, вычислительные алгоритмы часто таковы, что ошибка, возникшая на внутренней итерации, проявляется на граничной. Ниже будет показано на примере фортран-программ из набора NPB 2.3 [16], что покрытие операторов программы, собранное только на граничных итерациях циклов, практически не отличается от покрытия, собранного на всех итерациях.

[1] Институт прикладной математики им. М. В. Келдыша РАН, Миусская пл., 4, 125047, Москва; e-mail: krukov@keldysh.ru

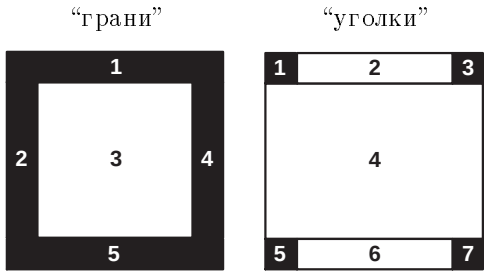
[2] Московский государственный университет им. М. В. Ломоносова, факультет вычислительной математики и кибернетики, Ленинские горы, 119992, Москва; e-mail: kudryavtsev@sp.cmc.msu.ru

Идея первого предложенного нами способа снижения требований на ресурсы памяти и времени для отладки заключается в сокращении количества контролируемых точек за счет сокращения количества анализируемых внутренних итераций — *контроль предлагается производить только на граничных итерациях циклов.*

Граничными итерациями ширины $k$ одномерного цикла являются $k$ первых и $k$ последних итераций этого цикла. Многомерными циклами будем считать только тесно вложенные одномерные циклы. Для многомерных циклов понятие границы можно ввести несколькими способами. Граничными итерациями типа *"грань" ширины* $k$ многомерного цикла будем считать те итерации, которые являются граничными ширины $k$ хотя бы для одного из одномерных циклов, входящих в состав заданного многомерного. Граничными итерациями типа *"уголки" ширины* $k$ многомерного цикла будем считать те итерации, которые являются граничными ширины $k$ для всех одномерных циклов, входящих в состав заданного многомерного. Параметр $k$ и тип границ можно выбрать экспериментально, оценивая покрытие операторов программы на предварительных запусках и выбрав подходящее.

Разбиение на граничные и внутренние итерации должно производиться отладчиком. Чтобы избавиться от необходимости на каждой итерации проверять, является ли она граничной или нет (а такая проверка отнюдь не тривиальна, поскольку надо учитывать границы циклов на каждом процессоре для обеих конфигураций процессоров, на которых были запущены сравниваемые программы), нужно разбивать итерации на блоки граничных и внутренних итераций. Отладчик должен обеспечивать правильный порядок обхода этих блоков для обеспечения корректного выполнения циклов с зависимостями по данным.

На рисунке приведены примеры разбиения множества итераций двумерного цикла с прямоугольным индексным пространством на граничные (черный цвет) и внутренние (белый) блоки. Внутренний одномерный цикл идет по столбцам, внешний — по строкам, обход столбцов производится слева направо, строк — сверху вниз. Цифрами обозначен порядок обхода блоков.




Для оценки эффективности предложенного метода граничных итераций и двойных тел циклов необходимо выбрать критерии. Такими критериями могут выступать: сокращение объема обрабатываемой информации; сокращение времени выполнения программ; потери по покрытию операторов отлаживаемой программы; потери по диапазону обнаруживаемых ошибок; вероятность пропуска ошибки; потери по точности определения отладчиком места ошибки (точности локализации).

Числовые характеристики по первым трем критериям приведены в п. 1.2. *По диапазону обнаруживаемых ошибок* в случае сравнительной отладки потерь не возникает, поскольку сравнительная отладка обнаруживает только различия в поведении и в данных сравниваемых программ, уменьшиться может только количество обнаруженных различий.

Ошибка "доступ к неинициализированным данным" может быть обнаружена анализом корректности, но поскольку анализ производится только на граничных итерациях, то множество элементов массивов и переменных, инициализируемых на внутренних итерациях, останутся для отладчика неинициализированными и при доступе к ним будут выданы ошибки. Поэтому при анализе корректности только на граничных итерациях необходимо учитывать возможность ложных диагностик об ошибках чтения неинициализированных данных или вообще отменить поиск этих ошибок.

*Оценки вероятности пропуска и точности локализации ошибок* сильно зависят от многих параметров — от программы и содержащихся в ней ошибок и от выбранных ширины и типа границ. Для адекватной практической оценки необходим представительный набор параллельных программ с типичными ошибками. Такого набора на сегодняшний день не существует. Вероятность пропуска и потери по точности локализации ошибок при использовании метода граничных итераций имеются как в случае динамического анализа корректности, так и при сравнительной отладке. Точность локализации ошибки при сравнительной отладке зависит от того, насколько быстро ее проявление отразится на граничных итерациях. Рассмотренный во втором разделе метод способен исключить пропуск ошибок и повысить точность их локализации при сравнительной отладке.

**1.2. Реализация и результаты экспериментов.** В качестве инструмента для реализации предложенных возможностей был выбран отладчик системы DVM, поскольку в нем уже были реализованы сравнительная отладка и динамический анализ корректности, для которых компиляторы обеспечивают

необходимую инструментацию исходного кода.

Мы разработали и реализовали в DVM-отладчике алгоритмы разбиения множеств итераций циклов с прямоугольным индексным пространством на граничные и внутренние блоки итераций. Для многомерных циклов реализованы два типа границ — "грани" и "уголки". Имеются возможности задания ширины границы. Ввиду большой зависимости от внутреннего устройства DVM-системы, алгоритмы разбиения циклов здесь не приводятся, но могут быть изучены по документации разработчика системы DVM [1].

Два вида отладочной инструментации исходного кода для метода граничных итераций (с одним телом цикла и с двумя телами циклов) были реализованы нами в системе DVM. Ниже представлены упрощенные примеры инструментации для одномерного цикла.

| без отладочной инструментации | инструментация "с одним телом цикла" | инструментация "с двумя телами цикла" |
|---|---|---|
| for I:=L to R do<br>{<br>  <тело цикла без<br>  вызовов отладчика><br>} | while (has_blocks( ... )) do<br>for I:=L1 to R1 do<br>{<br>  if (cond)<br>  <вызов отладчика><br>  <операторы><br>...<br>  if (cond)<br>  <вызов отладчика><br>  <операторы><br>} | while (has_blocks( ... )) do<br>if (cond)<br>  for I:=L1 to R1 do<br>  {<br>  <тело цикла с<br>  вызовами отладчика><br>  }<br>else<br>  for I:=L1 to R1 do<br>  {<br>  <тело цикла без<br>  вызовов отладчика><br>  } |

В инструментированные программы добавлены глобальная переменная логического типа cond и функция has_blocks, возвращающая значения логического типа. Эта функция производит разбиение множества итераций цикла на граничные и внутренние блоки итераций и возвращает значении "истина" столько раз, на сколько блоков был разбит исходный цикл. В нашем примере эта функция задает значения новым границам цикла — переменным L1 и R1 и переменной cond, значение которой истинно на граничных блоках итераций и ложно — на внутренних. В инструментации первого вида каждый вызов отладчика окружается условным оператором. В инструментации второго вида вместо одного тела цикла генерируется два: одно тело цикла — с вызовами отладчика, другое — без.

Параллельные многомерные DVM-циклы и соответствующие им циклы в последовательной программе при инструментации и работе отладчика рассматриваются как многомерные циклы. Вложенные последовательные циклы инструментируются и выполняются как вложенные одномерные циклы.

Сравнение эффективности вариантов инструментации проводилось на Фортран-DVM версиях программ пакета NPB 2.3 [16] класса A (набор из пяти параллельных ядер и трех модельных приложений, имитирующих вычисления и характеристики перемещения данных больших приложений гидрогазодинамики) на границах типа "уголки" ширины 1 на 16 процессорах. Имя класса определяет размеры основных массивов в приложениях NPB. Наш выбор класса A обусловлен желанием сравнить предложенные методы с существующими. На задачах бóльших размеров такое сравнение было бы невозможно.

В табл. 1 приведены средние значения по программам NPB для ЭВМ МВС-15000ВМ МСЦ РАН (процессоры PowerPC970 2.2 GHz, сеть Myrinet, оптимизация компилятора –O3 –qstrict).

Таблица 1

| среднее время | среднее отношение времен | | | | |
|---|---|---|---|---|---|
| без инстр. (сек.) | полная инстр./ без инстр. | инстр. 1 тело/ без инстр. | инстр. 2 тела/ без инстр. | полная инстр./ инстр. 1 тело | полная инстр./ инстр. 2 тела |
| 14,8 | 49 | 1,65 | 1,02 | 27 | 48 |

При полной отладочной инструментации вызовы отладчика производятся при каждой операции чтения или записи в переменную. Эта инструментация очень сильно замедляет выполнение программы, в

среднем в 49 раз. Программа, инструментированная с двойными телами и границей "уголки" ширины 1, практически не замедляется по сравнению с программой без отладочной инструментации: среднее замедление составляет 2 %. Инструментация с двойными телами заметно эффективнее инструментации с одним телом, в среднем более чем в 1,6 раза.

Эти результаты были вполне предсказуемы. Главными вопросами были два — удастся ли сократить трассы до приемлемого размера и насколько ухудшится при этом степень покрытия контролем операторов программы. Под степенью покрытия мы понимаем долю (в процентах) в числе тех операторов программы, которые выполнялись хотя бы один раз во время запуска программы с полной отладочной инструментацией. Для оценки степени покрытия в отладчике был реализован специальный режим работы. Пользователь может получить информацию о том, какие операторы его программы остались не покрыты при использовании граничных итераций. Следует отметить, что этот режим в дальнейшем планируется развить, чтобы находить минимальные множества итераций циклов, а не только граничных, обеспечивающих максимальное покрытие операторов программы (как при контроле всех итераций).

Таблица 2

| | 1 тело, полная инстр. | 2 тела, "уголки" ширины 1 | 2 тела, "уголки" ширины 2 | 2 тела, "грани" ширины 1 | 2 тела, "грани" ширины 2 |
|---|---|---|---|---|---|
| среднее покрытие операторов, % | 100 % | 99,4 % | 99,8 % | 99,8 % | 99,8 % |
| средний размер трасс, байт | 6,6 E+12 ($\sim$ 6 Tb) | 4,6 E+07 ($\sim$ 44 Mb) | 9,4 E+08 ($\sim$ 894 Mb) | 1,7 E+10 ($\sim$ 16 Gb) | 6,0 E+10 ($\sim$ 56 Gb) |
| среднее время генерации трасс, сек. | 27915 с. (7,75 ч.) | 15 | 19 | 105 | 287 |

В табл. 2 приведены оценки эффективности метода граничных итераций и двойных тел циклов по первым трем критериям из п. 1.1. В строках расположены средние значения покрытий операторов программы, размеров трасс и времени их генерации. При выполнении программ трассы генерировались, но не накапливались в памяти. В столбцах — различные типы инструментации программ. Использовались те же Фортран-DVM версии программ пакета NPB класса A, скомпилированные с оптимизацией компилятора. Запуски проводились на 16 процессорах МВС-15000ВМ.

Таблица 3

| имя теста | время выполнения | | | отношение времен | |
|---|---|---|---|---|---|
| | без инструментации (сек.) | с полным ДАК | с ДАК на "уголках" ширины 1 | ДАК полн./ ДАК на "угл." | ДАК на "угл."/ без инстр. |
| BT | 224,3 | не хватило памяти | 279,9 | — | 1,25 |
| CG | 7,5 | время $>$ 24 час | 9,4 | — | 1,25 |
| EP | 93,3 | 185945* | 95,8 | 1942 | 1,03 |
| FT | 16,8 | не хватило памяти | 27,1 | — | 1,61 |
| LU | 212,2 | 507494* | 224,9 | 2257 | 1,06 |
| MG | 7,6 | 19818 | 10,9 | 1823 | 1,43 |
| SP | 282,9 | 197215* | 320,6 | 615 | 1,13 |
| средн. | 120,7 | 227618 | 138,4 | 1659 | 1,25 |

Ниже в п. 2.2 приведены времена выполнения программ пакета NPB с реальным накоплением трасс и сравнением промежуточных результатов с данными, сохраненными в накопленных трассах.

В табл. 3 приведены данные о времени работы программ пакета NPB с динамическим анализом корректности (ДАК), запуски производились на одном процессоре. Некоторые их этих данных обозначены звездочками: это означает, что время выполнения соответствующего теста превышало максимально допустимое время работы программы в системе. Тест запускался с меньшим количеством итераций, и полученное время выполнения увеличивалось в соответствии с соотношением объемов вычислений.

Метод граничных итераций и двойных тел циклов продемонстрировал на границах ширины 1 и 2 для типов "грани" и "уголки" на 16 процессорах следующие результаты:

— сокращение размеров трасс в сотни и сотни тысяч раз (табл. 2);

— сокращение времени выполнения программ при генерации трасс и динамическом анализе корректности в сотни и тысячи раз (табл. 2 и 3);

— сохранение покрытия операторов программы составляет 99,4 – 99,8 % (табл. 2).

Однако данный метод имеет следующие недостатки:

— возможен пропуск ошибок, если они не проявились на граничных итерациях, либо выявление при сравнительной отладке далеких последствий этих ошибок;

— при динамическом анализе корректности необходимо или учитывать возможность появления ложных диагностик об ошибках "чтение неинициализированных данных", или вообще отменить поиск этих ошибок.

**2. Метод интегральных характеристик массивов. Комбинированный метод.**

**2.1. Описание метода.** Этот метод применим только к сравнительной отладке. Предлагается сравнивать не элементы массивов, в порядке доступа к ним на итерациях цикла, а только интегральные характеристики этих массивов (например, подсчитанные по какому-либо алгоритму контрольные суммы) по завершении циклов, в которых к этим массивам был доступ. Однако такое сокращение сравниваемой информации приводит к потере точности локализации ошибки — неизвестно, какой именно оператор внутри цикла привел к расхождению интегральных характеристик. Для точного указания ошибочного оператора мы модифицировали методику сравнительной отладки следующим образом.

1. Сначала отлаживаемая и эталонная программы выполняются со сравнением только интегральных характеристик. При их расхождении для какого-то массива пользователю выдается имя этого массива, динамическая точка, в которой обнаружено различие, и предшествующая точка, в которой интегральные характеристики этого массива еще совпадали (либо точка — начало программы, если подсчет интегральных характеристик этого массива производился впервые).

2. Далее, при повторном выполнении обеих программ производится подробная сравнительная отладка, но только на интервале, определяемом динамическими точками, полученными на предыдущем шаге. За пределами этого интервала сравнение данных вообще не производится.

Информацию о том, к каким массивам был доступ в цикле, во многих случаях можно получить с помощью статического анализа. Однако иногда такого анализа недостаточно, например при передаче массивов как параметров в процедуры или при использовании разных массивов в зависимости от параметра, и тогда необходим динамический анализ. Для повышения эффективности динамического определения используемых в цикле массивов и метода в целом можно производить отслеживание используемых массивов и вычисление интегральных характеристик только на граничных итерациях, используя разработанные алгоритмы из первого раздела статьи. Однако в этом случае есть риск пропуска обращений к тем массивам, к которым не было доступа на граничных итерациях, для этих массивов не будут вычислены и проконтролированы интегральные характеристики. Величина этого риска зависит от выбранных ширины и типа границ.

Недостаток метода, основанного на использовании интегральной информации, — потеря точности локализации ошибки в пределах одного или нескольких циклов — можно частично сгладить, если совместить подсчет интегральной информации со сравнением данных на граничных итерациях. Такой метод мы назвали *комбинированным*. Выгода его использования заключается в том, что если ошибка произошла на граничной итерации, то не потребуется повторных запусков для подробного сравнения.

Все перечисленные в п. 1.1 критерии оценки эффективности, кроме потерь по покрытию операторов программы, применимы к методу интегральных характеристик массивов. *Сокращение объема обрабатываемой информации и времени выполнения программ* могут быть оценены количественно, результаты приведены в следующем пункте. *Потерь по диапазону обнаруживаемых ошибок нет*, однако следует подчеркнуть, что этот метод неприменим для динамического анализа корректности.

*Вероятность пропуска ошибки* ничтожно мала, если данные должны полностью совпадать. В таком случае в качестве интегральной характеристики массива можно использовать контрольную сумму двоичных представлений элементов массива или несколько сумм, посчитанных по разным алгоритмам.

Если же полного совпадения ожидать не приходится, например, если запуски программы производятся на разных программно-аппаратных платформах, возникают серьезные проблемы и с выбором точности сравнения, и с выбором интегральных характеристик. Так, при использовании средних арифметических значений элементов массива можно не заметить существенных расхождений некоторых элементов. Необходимо отметить также, что при использовании метода интегральных характеристик совместно с методом граничных итераций возможна потеря массивов, для которых нужно было вычислять и сравнивать характеристики.

*Потери по точности определения места ошибки* ликвидируются за счет дополнительных запусков программы согласно приведенной выше методике. Без дополнительных запусков отладчик может определить только интервал, на котором возникло расхождение данных. При использовании комбинированного метода при проявлении ошибки на граничной итерации вызвавший ее оператор будет указан пользователю сразу.

**2.2. Реализация и результаты экспериментов.** Предложенный метод интегральных характеристик массивов реализован нами в DVM-отладчике с использованием контрольных сумм. Контрольные суммы вычисляются как средние арифметические элементов массивов. Реализована нумерация параллельных циклов и возможность задания точек начала и окончания сбора трассы в терминах этой нумерации. Реализован ряд опций, которые позволяют выбирать, для массивов какого типа (распределенных, обычных) и какого доступа (на чтение, на запись) вычислять контрольные суммы.

Таблица 4

| имя теста | время выполнения | | | |
|---|---|---|---|---|
| | без инструментации (сек.) | с накоплением подробной трассы | с накоплением трассы только с КС | с накоплением подробной трассы с КС |
| BT | 24,9 | 26,9 | 37 | 38 |
| CG | 2,1 | 2,2 | 2,2 | 2,2 |
| EP | 5,8 | 5,9 | 5,9 | 5,9 |
| FT | 2,9 | 3 | 3,3 | 3,3 |
| LU | 31 | 32,2 | 42 | 44,6 |
| MG | 1 | 1 | 1,2 | 1,2 |
| SP | 35,6 | 44,3 | 45 | 46,2 |
| средн. | 14,8 | 16,5 | 19,5 | 20,2 |

В табл. 4 приведены данные о времени выполнения программ пакета NPB с разными вариантами накопления трассы и сохранением ее во внешнюю память. В табл. 5 приведены размеры трасс для этих запусков. Программы были скомпилированы с двумя телами циклов и выполнялись на 16 процессорах ЭВМ МВС-15000ВМ с сохранением трассы и анализом доступа к массивам только на граничных итерациях типа “уголки” ширины 1. Контрольные суммы (КС) вычислялись по завершении параллельных циклов для тех распределенных массивов, к которым был доступ на запись внутри цикла. В табл. 6 приведены данные о времени выполнения программ пакета NPB на одном процессоре со сравнением промежуточных результатов с данными 16-процессорных трасс.

Накопление трасс с контрольными суммами (столбец 4 табл. 4) выполняется медленнее, чем накопление трасс на граничных итерациях без вычисления контрольных сумм (столбец 3 табл. 4). При сравнении трасс ситуация аналогична (столбцы 3 и 4 табл. 6). Причина этого замедления заключается в дополнительных межпроцессорных обменах при вычислении контрольных сумм. Максимальное замедление при накоплении трассы по сравнению со временем работы неинструментированной программы составило 53 % и было достигнуто на тесте BT (табл. 4). Максимальный размер трасс тоже достигнут на программе BT и составил примерно 160 мегабайт (табл. 5). Максимальное замедление при сравнении однопроцессорного выполнения программы с данными трассы достигнуто в программе MG и составило 49 % от времени выполнения неинструментированной программы (табл. 6).

На основе приведенных данных можно констатировать, что накопление трассы и сравнение запуска программы с трассой, собранной на угловых итерациях с вычислением интегральных характеристик массивов, незначительно замедляют выполнение программ пакета NPB класса A по сравнению с запусками

Таблица 5

| имя теста | размер подробной трассы (мегабайты) | размер трассы только с КС | размер подробной трассы с КС |
|---|---|---|---|
| BT | 160,68 | 1,67 | 160,78 |
| CG | 3,73 | 0,27 | 3,78 |
| EP | 0,49 | 0,01 | 0,49 |
| FT | 35,74 | 0,16 | 35,76 |
| LU | 66,42 | 1,48 | 66,53 |
| MG | 10,53 | 0,54 | 10,58 |
| SP | 31,07 | 0,63 | 31,14 |
| средн. | 44,09 | 0,68 | 44,15 |

Таблица 6

| имя теста | время выполнения без инструментации (сек.) | время сравнения с | | |
|---|---|---|---|---|
| | | подробной трассой | трассой только с КС | подробной трассой с КС |
| BT | 224,3 | 274,3 | 281,1 | 286,9 |
| CG | 7,5 | 7,7 | 8 | 8,4 |
| EP | 93,3 | 93,8 | 93,9 | 93,9 |
| FT | 16,8 | 18,1 | 19,9 | 20,5 |
| LU | 212,2 | 226,3 | 232,2 | 238,8 |
| MG | 7,6 | 7,9 | 11,3 | 11,4 |
| SP | 282,9 | 286 | 290,9 | 296,1 |
| средн. | 120,7 | 130,6 | 133,9 | 136,6 |

без отладки, делая возможным применение сравнительной отладки к реальным приложениям. При этом размеры трасс вполне приемлемы для практического использования предложенных методов граничных итераций, двойных тел циклов и интегральных характеристик массивов.

**3. Метод коррекции результатов редукционных операций.** Сравнительную отладку невозможно использовать при недетерминированном поведении программ. Серьезные трудности возникают и тогда, когда нельзя исходить из точного совпадения данных, например в случае сравнения результатов выполнения программы на разных ЭВМ. Величина допустимых расхождений данных при двух запусках программы зависит как от ЭВМ, на которых производились запуски, так и от самой программы. Кроме того, расхождения могут накапливаться в процессе выполнения сравниваемых программ. При сравнении с большой точностью возможна выдача множества несущественных расхождений, а при выборе низкой точности сравнения возможен пропуск первого существенного расхождения.

Если ограничиться случаем использования одной ЭВМ (а именно такая ситуация типична для отладки и параллельных, и последовательных программ), то проблемы точности сравнения нет. Для обеспечения детерминизма отлаживаемой программы можно воспользоваться методом коррекции ее поведения в соответствии с поведением эталонной программы. Это можно делать без какого-либо участия программиста в том случае, когда источником недетерминизма являются те или иные конструкции языка или функции библиотеки параллелизма. Коррекция поведения отлаживаемой программы предполагает наличие эталонной трассы или опережающее выполнение эталонной программы. Например, типичным источником недетерминизма в MPI-программах [17] является возможность при приеме сообщения не указывать процесс-отправитель. Это может привести к различиям в поведении программы в зависимости от того, от кого пришло сообщение. Однако если известно, от кого пришло сообщение в эталонной программе, можно в отлаживаемой программе скорректировать функцию приема сообщения, задав явно того же отправителя.

В DVM-программах практически единственным источником недетерминизма, за исключением функций опроса времени и получения случайных чисел, является вычисление значений редукционных сумм и произведений. Этот недетерминизм вызван изменением порядка выполнения редукционных операций сложения и умножения при параллельном выполнении. Однако обеспечение порядка выполнения этих операций, эквивалентного их порядку при последовательном запуске, приведет к существенному замедлению параллельной программы. Этого замедления можно избежать, если корректировать не процесс вычисления редукции при параллельном запуске, а лишь ее результат.

Мы реализовали в DVM-отладчике возможность замены вычисленных редукционных значений значениями эталонной трассы. Однако такая замена производится не всегда, а только в тех случаях, когда вычисленные значения отличаются от эталонных в пределах заданной пользователем точности. Предложенный метод назван *методом коррекции результатов редукционных операций.*

Были проведены эксперименты с Фортран-DVM программами пакета NPB класса S. Класс S был взят для простоты, размеры массивов в нем меньше, чем в классе A. При сравнении однопроцессорных запусков с двух- и четырехпроцессорными без использования коррекции результатов редукционных операций во всех программах пакета были обнаружены многочисленные различия данных. При использовании коррекции результатов редукционных операций различий не обнаружено ни в одной программе. В случае запуска ошибочной программы с опцией коррекции редукционных результатов и сравнением данных на полное совпадение мы сразу бы увидели первое существенное различие (кроме множества несущественных, вызванных недетерминизмом).

**Выводы.** В настоящее время остро ощущается отсутствие развитых и автоматизированных средств отладки, способных существенно облегчить и ускорить разработку параллельных программ. Такие средства могли бы быть построены на базе новых подходов к отладке параллельных программ, допускающих автоматизацию, — сравнительной отладки и динамического анализа корректности, однако эти подходы предъявляют высокие требования к ресурсам памяти и времени и поэтому неприменимы к реальным научно-техническим приложениям. Кроме того, сравнительную отладку невозможно использовать при недетерминированном поведении программы.

В настоящей статье мы представили новые результаты наших работ, направленных на создание развитых и автоматизированных средств отладки параллельных программ.

Разработаны новые методы автоматического обнаружения некорректного выполнения параллельных программ, написанных в моделях параллельного программирования с глобальным адресным пространством (OpenMP [18], HPF [19], DVM [20]). Эти методы, базирующиеся на динамическом контроле выполнения каждого оператора программы посредством его сопоставления со спецификациями параллелизма или с поведением эталонной программы, способны существенно упростить и ускорить отладку параллельных программ.

Разработаны алгоритмы автоматического определения контролируемых точек выполнения программы, обеспечивающие существенное сокращение требуемых ресурсов времени и памяти и высокую точность

обнаружения первых проявлений некорректного поведения параллельной программы. Разработанные методы и алгоритмы автоматического обнаружения некорректного выполнения параллельных программ реализованы в отладчике DVM-программ. Проведено экспериментальное исследование эффективности созданного отладчика на представительном наборе параллельных программ, подтвердившее высокую эффективность разработанных методов и алгоритмов.

СПИСОК ЛИТЕРАТУРЫ


1. Сайт системы DVM. Документация пользователя и разработчика (www.keldysh.ru/dvm).
2. *Алексахин В.Ф., Ефимкин К.Н., Ильяков В.Н., Крюков В.А., Кулешова М.И., Сазанов Ю.Л.* Средства отладки MPI-программ в DVM-системе // Научный сервис в сети Интернет: Труды Всероссийской научной конференции. М.: Изд-во МГУ, 2005. 113–115.
3. *Vetter J.S., de Supinski B.R.* Dynamic software testing of MPI applications with Umpire // Proc. SC2000: High Performance Networking and Computing Conf. Dallas(TX,USA), 2000 (http://www.llnl.gov/CASC/people/vetter/pubs/sc00-umpire-vetter.pdf).
4. MPI-CHECK (http://andrew.ait.iastate.edu/HPC/MPI-CHECK.htm).
5. Marmot (http://www.hlrs.de/organization/amt/projects/marmot/).
6. *DeSouza J., Kuhn B., de Supinski B.R.* Automated, scalable debugging of MPI programs with Intel Message Checker // Proc. of the Second International Workshop on Software Engineering for High Performance Computing System Applications. New York: ACM Press, 2005. 78–82 (http://csdl.ics.hawaii.edu/se-hpcs/papers/11.pdf).
7. HP Visual Threads (http://h18000.www1.hp.com/products/software/visualthreads).
8. Intel Thread Checker (http://www.intel.com/cd/software/products/asmona/eng/threading/286406.htm).
9. *Крюков В.А., Удовиченко Р.В.* Отладка DVM-программ // Программирование. 2001. № 3. 19–29.
10. *Крюков В.А., Удовиченко Р.В.* Отладка DVM-программ. Препринт ИПМ им. М.В. Келдыша РАН № 56. М., 1999.
11. *Abramson D.A., Sosic R.* Relative debugging using multiple program versions // Intensional Programming I. Sydney: World Scientific. 1995 (http://www.csse.monash.edu.au/%7Edavida/papers/islip.pdf).
12. *Manne F., Andersen S.O.* Automating the debugging of large numerical codes // Modern Software Tools for Scientific Computing. Cambridge (MA, USA): Birkhauser Boston Inc., 1997. 339–352 (http://www.ii.uib.no/ fredrikm/fredrik/papers/debug.ps).
13. *Hood R., Jost G.* Support for debugging automatically parallelized programs // Proc. of AADEBUG'2000. Munich, 2000 (http://arxiv.org/ftp/cs/papers/0012/0012006.pdf).
14. *Matthews G., Hood R., Johnson S., Leggett P.* Backtracking and re-execution in the automatic debugging of parallelized programs // Proc. of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11'02). Washington (DC, USA), 2002 (http://csdl.computer.org/comp/proceedings/hpdc/2002/1686/00/16860150abs.htm).
15. *Matthews G., Hood R., Jin H., Johnson S., Ierotheou C.* Automatic relative debugging of OpenMP programs. NAS Technical Report NAS–03–014. Moffett Field (CA, USA), 2003 (http://www.nas.nasa.gov/News/Techreports/2003/PDF/nas-03-014.pdf).
16. NAS Parallel Benchmarks (http://www.nas.nasa.gov/Software/NPB/).
17. Message-Passing Interface Forum (http://www.mpi-forum.org).
18. OpenMP Consortium (http://www.openmp.org).
19. High Performance Fortran Forum (http://www.hipersoft.rice.edu/hpff/).
20. *Коновалов Н.А., Крюков В.А., Михайлов С.Н., Погребцов Л.А.* Fortran-DVM — язык разработки мобильных параллельных программ // Программирование. 1995. № 1. 49–54.


Поступила в редакцию
16.09.2006