УДК 519.6

РАЗРАБОТКА СРЕДСТВ ВИЗУАЛИЗАЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ. ВИЗУАЛЬНОЕ ПРОГРАММИРОВАНИЕ И ВИЗУАЛЬНАЯ ОТЛАДКА ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

В. Л. Авербух, А. Ю. Байдалин
(ИММ УрО РАН)

Работа содержит обзор состояния дел в визуализации программного обеспечения параллельных вычислений, касающийся проблем визуального программирования и визуальной отладки параллельных программ. Рассмотрены также результаты разработки сред поддержки программирования и отладки параллельных программ для системы DVM.

Визуализация программного обеспечения параллельных вычислений включает в себя исследования и разработку визуальных языков параллельного программирования, визуальных отладчиков правильности, а также систем настройки, отладки, измерения и анализа производительности параллельных программ. Если в случае последовательных систем визуализация программного обеспечения (Software Visualization) играет зачастую лишь вспомогательную роль, то в параллельных вычислениях она, как правило, оказывается жизненно необходимой.

Прежде всего это связано с отсутствием систем параллельного программирования, позволяющих эффективно автоматизировать процессы программирования и отладки в случае масспараллельных вычислений и тем самым понизить их интеллектуальную сложность. Как следствие, литература по данному вопросу заполнена жалобами на проблемы параллельного программирования и утверждениями, что только средства визуализации программного обеспечения могут помочь при разрешении стоящих перед программистами задач. Действительно, визуализация служит средством облегчения тех сложностей, которые несет параллельное программирование, и особенно отладка параллельных программ.

Вторая задача визуализации — обеспечение эффективности (лучшей производительности) параллельных программ. Поэтому значительная часть проектов, вышедших за рамки академических, относится к системам отладки, настройки, измерения и анализа производительности параллельных программ. Большинство этих систем обеспечивает сбор данных и отображение метрик производительности параллельных программ. Отметим, однако, что при этом основные проблемы возникают не столько в процессе поиска причин неэффективности, сколько при выяснении того, что нужно сделать, чтобы программа заработала с должным ускорением.

Проблематика визуализации программного обеспечения параллельных вычислений освещается в литературе начиная со второй половины 80-х годов. Опубликованы сотни работ, описывающих конкретные системы визуализации, а также статьи, содержащие интересные обзоры и сравнительный анализ систем, например [1—4] и др. На русском языке за последние десять лет было опубликовано несколько работ авторов данной статьи, также содержащих обзоры состояния дел в визуализации параллельных вычислений [5—8].

В данной статье приводится краткий анализ идей, заложенных при проектировании систем визуализации программного обеспечения параллельных вычислений, по разделам, включающим средства визуального параллельного программирования и средства отладки правильности параллельных комплексов. Кроме того, рассматриваются авторские разработки средств визуализации для нужд отечественной системы параллельного программирования DVM [9]. В

следующей работе предполагается опубликовать материалы по отладке, настройке и анализу производительности параллельных программ.

Визуальные языки параллельного программирования

Разработка визуальных языков параллельного программирования началась еще в 80-е годы. Изучение разработанных систем позволяет выделить несколько подходов к визуализации.

Один из подходов предусматривает описание взаимодействия параллельных процессов с использованием уже имеющегося опыта создания диаграмматических и иконических языков и имеющихся графических нотаций. При этом ставятся задачи — обеспечить отображение таких понятий, как посылка, порт, одновременное исполнение, или при задании общей структуры системы описать наличие иерархичности, перехода из состояния в состояние и т. п. То есть предусматривается полное визуальное задание всех (или большинства) понятий и получение визуального аналога текстовых сред параллельного программирования, в которых пользователь должен, в частности, подробно описывать посылку сообщений, порты ввода/вывода и структуру процессов, а также обеспечивать работоспособность и эффективность параллельной программы.

Идеи использования диаграмм для представления взаимодействующих процессов, примененные в одном из первых визуальных языков параллельного программирования Pigsty/I-PIGS [10], получили развитие по нескольким направлениям.

В рамках этого подхода наиболее развитым представляется язык GRAPNEL [11], использующий для описания полноценной распределенной прикладной программы как графическое, так и текстовое представление. Графика применяется только для того, чтобы изобразить те части программы, которые содержат важные элементы параллелизма и взаимодействия процессов, а текстовые описания — например, для декларации и определения данных, сегментов программы, не содержащих обменов, и т. д. В языке реализована графическая поддержка структурированного проектирования на уровне процессов, динамического создания и уничтожения процессов; предопределены топологии схем (например двумерная сетка, дерево, кольцо и т. п.), которые могут использоваться для установки регулярной топологии процессов. При этом размер топологий определяется во время выполнения. Именно эти идеи особенно полезны для дальнейшего развития визуальных языков параллельного программирования. При развитии системы предполагалась также поддержка высокоуровневой отладки. Та же самая графическая среда должна использоваться для того, чтобы и писать, и отлаживать прикладную программу.

Как и в последовательных языках, в параллельных также имели место попытки создания истинно визуальных языков, таких, что не требуют текстового представления, а позволяют программистам (или заставляют их) отобразить свое двумерное понимание задачи непосредственно в вычисления.

В рамках системы VISO [12] реализован практически полный визуальный вариант языка параллельного программирования Occam. В составе его словаря — набор иконов (пиктограмм), представляющих все программные конструкции Occam'a. Графический синтаксис также базируется на языке Occam.

Следует отметить, что использование искусственной образности не увеличивает выразительности нового языка, но зачастую затрудняет понимание языковых конструкций.

При другом подходе также используется графовое представление потока управления и/или потока данных. В визуальном виде отображается и распараллеливание процессов (или потенциальное распараллеливание). Например, граф потока данных может показывать создание данных одним последовательным вычислением для нужд другого. Графически программа выглядит как привычный (последовательный) граф потока данных. Вычисления традиционно задаются узлами, параллельность — дополнительной (параллельной) дугой или специальным значком.

Отметим, что при этом подходе обеспечение эффективности параллельной программы возлагается на компилятор (или эта проблема просто игнорируется).

В ряде систем параллельного программирования, разработанных в первой половине 90-х годов, например в системе на базе языка Phred [13] (который не только является языком описания вычислений, но и может описывать детерминированное поведение параллельных программ), описание взаимодействия процессов вообще оказывается не важно .

Во второй половине 90-х кроме языков, использующих различные комбинации графов потока управления и потока данных (например, язык PCG [14]), появились визуальные языки параллельного программирования на базе сетей Петри [15] и других визуальных графовых формализмов.

В языке Vorlon [16] реализована модель *параллельного исполнения потока объектов* (parallel object-flow execution model). Эта модель объединяет объектно-ориентированную модель и модель потока данных. Таким образом, существует возможность обеспечить задание как параллелизма, так и всей сложности конкретной прикладной задачи. При этом такие аспекты параллелизма, как синхронизация, обеспечиваются за счет конструкций, поддерживающих поток данных, а задание типов и их взаимосвязи дает возможность описывать все многообразие прикладной области. Параллельный граф потока объектов состоит из узлов и дуг. Узлы представляют некоторую форму вычисляемого элемента, а дуги описывают взаимозависимости потоков управления и возможные маршруты передачи параметров между узлами.

Также в 90-е годы были сделаны попытки реализации такой естественной идеи, как построение визуальных языков на базе методик отображения параллельности, ранее использованных в системах отладки и настройки эффективности параллельных программ.

В языке визуального параллельного программирования Visper [17] введено понятие *графа взаимодействия процессов*, которое, в свою очередь, опирается на понятия *карта параллельности* и *диаграмма пространства-времени*.

Карта параллельности [18] отображает возможную параллельность задачи. Это одновременно структура данных для перезапуска (переигровки) потока управления и графический метод представления параллельных процессов. Карта показывает историю проработки процессов в виде потока событий на временной шкале. Каждая колонка шкалы показывает последовательный поток событий одного процесса. Строка шкалы представляет временной интервал. Все события, которые появляются в позициях этой строки, могут произойти одновременно. Данный метод визуализации параллельных и распределенных процессов может быть использован при мониторинге, отладке и анализе эффективности систем рассматриваемого типа.

Динамика выполнения параллельной программы в диаграммах пространства-времени представляется как поток событий в двумерном пространстве, где одна ось показывает время, а вторая — реальные отдельные процессоры.

Программирование на языке Visper не является полностью визуальным. Визуальные механизмы широко используются в тех случаях, когда обнаружены потоки управления и параллелизма. Декларативные и последовательные фрагменты программного кода пишутся на обычных языках, таких как C или Fortran, и, таким образом, не представляются в виде визуальных компонентов.

Еще один подход к разработке визуальных языков описания параллельных вычислений предусматривает естественное (в смысле прикладной области) представление самих математических объектов, а не графовое представление потока управления или потока данных соответствующей программы. При этом в ряде случаев делается попытка визуализации как параллелизма, так и динамики обработки данных [19]. Предусматривается прямое отображение таких визуальных спецификаций непосредственно в программу конкретной ЭВМ. Визуальные образы представляют высокоуровневые математические объекты, например параметризированную матрицу и вектор при описании методов решения задач линейной алгебры. В других случаях [20] визуальный интерфейс, задающий начальные значения для прикладной вычислительной системы, можно рассматривать как специализированный визуальный язык параллельного программирования. Эффективность распараллеливания обеспечивается специализированными средами программирования, а не описывается явно пользователем.

Наконец, отметим очень интересную работу [21], в которой описывается использование средств визуализации для распараллеливания программ, а также для увеличения параллелизма циклов.

В этой работе вводится понятие *графа зависимостей итерационного пространства*. Узлы этого ациклического графа представляют итерации цикла, а дуги — зависимости между итерациями. Он показывает точные зависимости данных произвольно вложенных циклов. Граф (в трехмерном виде) строится на основе полученной средствами системы трассы програм-

мы. Различные графические операции, такие как поворот, наплыв (zooming), фрагментирование, раскраска, фильтрация, примененные к графу, позволяют детально исследовать отношения зависимости. Далее анимированный граф потока данных выполнения программы показывает максимальный параллелизм, а параллельные циклы выявляются автоматически путем встроенных средств анализа зависимостей.

Система, построенная на базе графа зависимостей итерационного пространства, была успешно опробована на реальной задаче.

Еще раз отметим, что в большинстве визуальных языков для представления понятий параллельного программирования используется искусственная образность, освоение которой требует определенных усилий от пользователя. Зачастую визуальные системы программирования навязывают пользователю не только новую графическую (графовую) нотацию, но и новую технологию программирования. Опыт авторов показывает, что надо исходить из практики пользователя и поддерживать в визуальных средах уже сложившиеся приемы и методы параллельного программирования.

В [8] рассмотрены некоторые решения, использованные авторами при проектировании средств визуальной разработки для системы параллельного программирования DVM [9].

При распараллеливании DVM-программы от пользователя (программиста-прикладника) требуется точное понимание и описание ряда моментов, по которым в последовательном случае вопросов не возникает. Специфика разработки DVM-программ заключается в распараллеливании по данным. Кроме того, существенным является наличие технологии программирования на DVM, предполагающей написание (и отладку правильности) DVM-программы в однопроцессорном варианте (на ПЭВМ пользователя), а затем перевод ее на параллельный вычислитель.

Параллелизм по данным означает, что в параллельной программе одни и те же действия осуществляются одновременно над множеством наборов данных. В DVM такими наборами данных являются фрагменты массивов, распределенные по виртуальным процессорам, а действиями — тела параллельных циклов или фрагментов задачи, являющиеся, в конечном итоге, наборами операторов на процедурных языках C и Fortran.

Для распараллеливания от пользователя требуется четкое представление о функционировании программы и ее работе с данными. Необходимо ответить на ряд вопросов, в том числе:

- какова требуемая пользователем топология виртуальных процессоров;
- как распределяются по процессорам массивы (это влияет на загрузку процессоров и, в конечном счете, на время работы программ);
- как организованы циклы с точки зрения обработки элементов массивов;
- какие данные нужны для тех или иных вычислений;
- какие существуют зависимости между данными по очередности вычисления.

Распараллеливание начинается с ответа на этот ряд четких вопросов и заканчивается точным описанием всего необходимого в терминах DVM-директив. Поэтому было предложено использовать не абстрактные графические образы, а диалоговые формы, аналогичные стандартно применяемым в различных системах тестирования и анкетирования. При этом исходный текст генерируется на основе информации, заданной пользователем. Таким образом, можно говорить о кодировании с использованием *мастера текста* (рис. 1.)

В рассматриваемом случае мастером текста является форма, при взаимодействии с которой пользователь отвечает на вопросы путем выбора элемента из списка или указания конкретного имени. По окончании работы с мастером пользо-

Рис. 1. Схема мастера текста для генерации директивы DISTRIBUTE


Распределение массивов
Язык
DVM-C
DVM-Fortran
Тип распределения
Distribution Порючков
Redistribution Вторичное
Размерность
1
Распределяемая размерность
список распределений:
распределение каждой
размерности
Массивов
1 (один)
N (много)
Имя/Список имен
имена массивов
Тип распределения
* (Локально)
BLOCK (Блок)
GEN_BLOCK (Блок с весом)
Массив весов
имя массива
На массив процессоров
имя массива процессоров
Окно результата – текст DVM директивы
Создать
Очистить
Копировать
Отмена

ватель получает текст, сгенерированный на основе указанной им информации. Текстовые мастера используются для генерации DVM-команд, а также для задания ряда DVM-директив, требующих больших по объему спецификаций. К примеру, при описании распараллеливания цикла необходимо явно указать все используемые переменные, размещенные на других процессорах.

Использование диалоговой схемы взаимодействия и привычных элементов управления не требует коренного изменения мышления или дополнительных навыков. Применение текстовых мастеров для разработки параллельных программ не отменяет написание кода пользователем, но заметно облегчает этот процесс. Также текстовые мастера могут использоваться в качестве связующего звена между редактором исходного текста и графическими схемами проектирования (скрывающими получаемый исходный текст).

Система визуальной поддержки проектирования в какой-то мере превращается в визуальную систему программирования, где одним из главных видов отображения будет представление исходного текста исходным же текстом, из окна редактирования которого вызываются мастера (для непротиворечивого изменения). От традиционных визуальных систем, требующих рисования диаграмм, такая концепция отличается большей понятностью для человека, привыкшего к программированию на уровне исходного текста. По сути мастера текста могут считаться узкоспециализированными системами визуального программирования на основе SpreadSheet (большой таблицы).

Одним из факторов эффективного распараллеливания является удачный выбор топологии процессоров. Существуют примеры систем, предусматривающих настройку физической топологии процессоров под конкретную задачу [22]. В других случаях предусмотрена настройка логической топологии. В визуальном виде такая настройка реализована в GRAPNEL. В общем виде конфигурирование логической топологии предусматривается в коммуникационных библиотеках, например в MPI.

В DVM введена концепция виртуальных процессоров, на массив которых происходит распределение данных (фрагментов многомерных массивов). На виртуальных процессорах происходит выполнение витков параллельных циклов и фрагментов задачи. Под фрагментом задачи подразумевается некоторый блок операторов, помеченный специальными директивами и выполняющийся параллельно с другими такими же фрагментами. Например, фрагментами задачи могут быть вызовы некоторых подпрограмм. Одномерный массив данных может быть распределен по одномерному массиву процессоров одним из нескольких способов: равномерно, блоками равной длины или блоками переменной длины. В многомерных массивах распределению может подвергаться каждое измерение. Кроме того, для эффективности вычислений бывает полезно обеспечить *выравнивание* — распределение на один процессор определенных элементов различных массивов.

Для облегчения проектирования программ и отладки правильности были предприняты попытки разработать средства визуализации распределения данных. Средствами визуального распределения нужно было решать по сути две противоположные задачи: по взаимодействию с графическим образом генерировать исходный текст, а данные о распределении, имеющиеся в исходном тексте, в свою очередь, отображать на графической схеме. В основе *визуального распределителя* лежит принцип непосредственного действия, когда манипуляции с графическим объектом приводят к изменению исходного текста, в данном случае модификации DVM-директивы DISTRIBUTE. В качестве графического объекта выступает таблица, представляющая вектор или двумерный массив данных. Распределение ("распиливание") массива по процессорам изображается более жирными линиями, разграничивающими ячейки-элементы. По выбору элемента массива во "всплывающей" подсказке и специальном окне на форме визуального распределителя выводится информация о процессоре, на который он назначается. Экспериментальные прототипные системы показали возможность решения задачи визуализации распределения для одно- и двумерных массивов (рис. 2.)

Наибольший интерес (и трудности у пользователей) представляет работа с многомерными массивами. Однако отображение таких объектов уже само по себе является задачей очень нетривиальной [23]. Еще более сложная задача — разработка взаимодействия пользователя с такими объектами. Однако принцип указания распределения для каждого измерения позволяет декомпозировать многомерное распределение в совокупность одно- и двумерных с последующей взаимосвязанной их визуализацией.

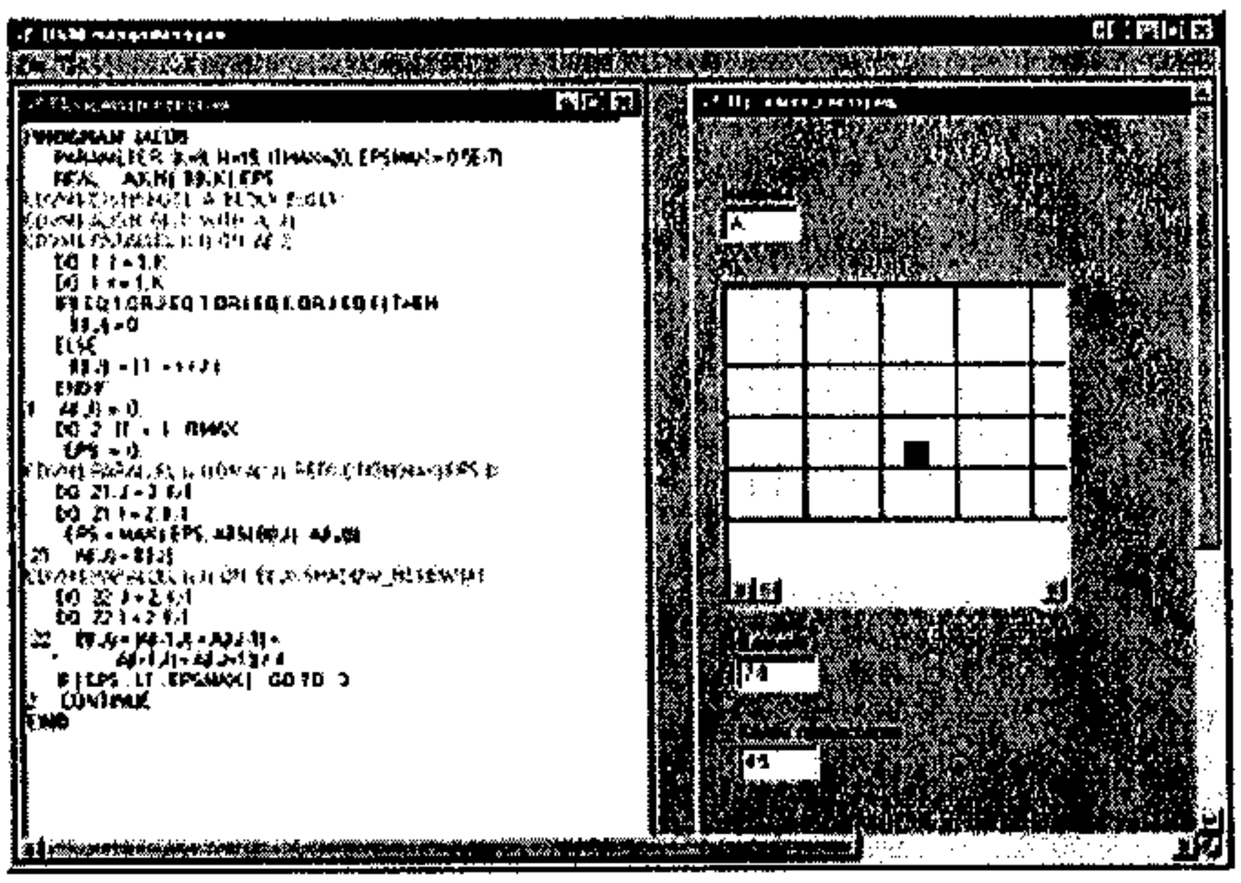

Рис. 2. Визуализатор распределения массива по процессорам

Развитие идей, использованных в средствах визуализации разработки DVM-программ, позволяет перейти к проектированию более универсальных сред, пригодных и для MPI-программ. По мнению авторов, следует продолжить развитие систем визуальной поддержки проектирования и программирования, опираясь на имеющиеся технологии разработки параллельных программ, а не навязывать пользователю новые методики работы.

Очевидно, чисто визуальные методы программирования отдельных процессов для случаев программирования на основе обмена сообщениями не подходят для разработки серьезных программ. В частности, текстовое программирование обеспечивает большую гибкость и лучшую переносимость кодов, чем визуальное. Не ясно также, следует ли уделять большое внимание визуальному описанию самого процесса передачи сообщений. Однако полезными представляются такие методы поддержки разработки, как использование мастеров текстов и возможных директив распараллеливания.

На следующем этапе разработки визуальных сред параллельного проектирования необходимо создать технологию, опирающуюся на сочетание текстового программирования (для описания процессов и операций взаимодействия между ними) с мастерами текстов и директив. Из методик настоящего визуального программирования полезным представляется прежде всего использование встроенных и масштабируемых шаблонов, аналогичных примененным в GRAPNEL. В рамках этой технологии пользователь может начать разработку модулей в некоторой текстовой среде программирования. Описание взаимодействия процессов возможно осуществить, используя соответствующие мастера портов, операций ввода/вывода и коллективного взаимодействия. Встроенные шаблоны (ферма, линейный массив, кольцо, сетка, дерево и т. п.) помогут пользователю определять необходимые топологии параллельной задачи.

В ряде визуальных систем параллельного программирования проявляется принцип единства видов отображения, предназначаемых как для визуального проектирования и кодирования, так и для отладки программ. Еще интереснее попытки использовать для спецификации параллельных программ методики, применяемые в системах отладки и настройки эффективности. Кроме упомянутых выше карт параллельности и диаграмм пространства-времени в языке Visper, возможно использование в рамках некоторого динамического визуального языка параллельного программирования на базе аналога диаграмм Гантта, которые традиционно применяются в системах отладки производительности параллельных программ. Например, непосредственное воздействие пользователя на элементы диаграмм, которые отражают времена, соответствующие тому или иному состоянию процессов (активный, простой, ожидание, обмен), должно изменять реальные временные характеристики разрабатываемого процесса.

Наибольший интерес представляют попытки разрабатывать системы программирования, которые могут предупреждать программиста о потенциальной недетерминированности и других потенциальных ошибках, помогая также при отладке параллельных программ, как это делает система визуального программирования Phred. По мнению авторов, при разработке сред визуального программирования параллельных вычислений необходимо обеспечение общего подхода к визуализации при поддержке всего цикла *проектирование — редактирование — выполнение — отладка — отладка эффективности.*

Визуальная отладка параллельных программ

При отладке параллельных программ возникают новые (по сравнению с последовательными программами) классы ошибок, связанные, во-первых, с множеством отдельных потоков управления (процессов), которые выполняются одно-

временно, а во-вторых, с асинхронными взаимодействиями. Кроме базовых текстовых отладчиков на отдельных процессорах вычислительного комплекса, большой эффект обеспечивают визуальные средства отладки, представляющие работу прикладной программы, выполняемой на параллельном вычислителе.

Интерактивная отладка предполагает остановку выполнения программы в интересующем программиста месте, изучение и, возможно, изменение ее состояния. Для параллельной программы "интересные точки", как правило, описываются в терминах событий, которые произошли во время ее выполнения на различных процессорах. Распознавание интересных точек может основываться на анализе, проведенном на различных процессорах при помощи специально размещенных для этой цели программ. В другом случае этот анализ может быть осуществлен на какомлибо процессоре, внешнем по отношению к данному набору. Тогда для того, чтобы направить сообщение о возникших событиях, данные о них посылаются на внешний процессор.

Визуализация программирования часто включает в себя динамический вывод поведения программы. Вывод управляется посредством отслеживания интересных изменений состояния системы в соответствии с выполнением программы. Для параллельных систем эти изменения состояний могут резко возрастать в связи с комбинацией событий, происшедших на многих различных процессорах. Часто в случае визуализации каждый процессор параллельной системы отправляет поток записей, описывающих эти события, на центральную систему визуализации.

Визуализация может осуществляться *посмертно*, т. е. после завершения выполнения программы и сбора всех данных о событиях, или может быть *живой* (on-line), осуществляемой параллельно с выполнением программы по мере получения информации о событиях.

Живая визуализация программирования существенна для объединения визуализации и отладки. Системы визуализации показывают поведение программы по мере получения данных о событиях, а отладчики позволяют подробно изучать состояния программы, переходя от одной контрольной точки к другой при управляемых ее перезапусках. Живая визуализация предпочтительна также при мониторинге хода работы программы, выполняемой продолжительное время на большой вычислительной системе, как для контроля правильности работы программы, так и для выявления точек, в которых возникают проблемы. Существенным компонентом живой визуализации является непрерывное (немедленное) упорядочение и компоновка поступающей информации о событиях.

Отладка параллельных программ считается более сложной, чем традиционная последовательная отладка.

Первая причина заключается в том, что имеется множество отдельных потоков управления (процессов), которые выполняются одновременно. Поэтому невозможно не только идентифицировать ошибочное выражение или команду, но даже указать процесс, содержащий это ошибочное выражение. Это то, что называется отладкой на уровне процессов.

Поскольку процесс асинхронно взаимодействует с другими процессами, задача отладки еще больше усложняется. Действительно, такое взаимодействие не только вызывает новый класс ошибок (таких как *гонки* или тупиковые ситуации), но также способствует быстрому распространению ошибки на всю вычислительную систему (так называемый *эффект домино*). Тем не менее, если процесс не выполняет то, что должен, возможно, причина этого — в получении процессом неправильных данных, а не в наличии в нем ошибки.

Одна и та же параллельная программа при множестве успешных запусков, имея на входе одни и те же данные, может получить и различную последовательность состояний, и различные результаты. Эти расхождения вызываются тем, что программа может содержать *гонки сообщений* (data races). События, такие как получение сообщений, приходящих почти одновременно, могут вызвать различный порядок успешного выполнения программы, а программа может получить различие в результатах. Повторение симптомов гонки сообщений может казаться затруднительным, так как добавление кодов для более полного изучения поведения программы может увеличить время ее работы и тем самым нарушить условия появления гонки.

Существуют две методики решения этой проблемы — введение *контрольных точек* и *регистрация событий.*

В системах, основанных на контрольных точках, процессоры периодически записывают все свои состояния или их часть во внешнюю память. Перезапуск программы из контрольной точки в течение последующих запусков выну-

ждает ее вернуться к состоянию, которое она имела в контрольной точке при первоначальном выполнении. Системы, основанные на контрольных точках, удобны для программиста при перезапуске длинных программ с середины, но генерируют слишком много данных.

Системы, основанные на регистрации событий, отказываются от пошагового повторения и, таким образом, не нуждаются в хранении специальной информации о состоянии программы.

Системы с перезапуском позволяют гарантировать только то, что программа повторяема с начала до конца, а фиксация трассы включает в себя лишь порядок межпроцессорных событий, но не их содержание. Оба типа систем, основанных как на контрольных точках, так и на регистрации событий, стараются минимизировать свое влияние на время выполнения программы, но, конечно, полностью избежать этого не могут.

Гонка сообщений может все же вызывать недетерминированное поведение, когда пользователь пробует фиксировать конкретный образец ошибки. Ошибка фиксируется только один раз, хотя пользователь может неоднократно перезапускать программу через отладчик, обычно предоставляющий более серьезные средства отладки, чем простой оператор print. Другая важная проблема, с которой сталкивается каждый, кто отлаживает параллельные программы, заключается в асинхронных взаимодействиях, которые могут как произойти, так и не произойти, и тем самым чрезвычайно затрудняют повторное выполнение программы и препятствуют восстановлению ошибочной ситуации, создавая недетерминированность в поведении программы. Ввод в тело отлаживаемой программы дополнительных операторов для получения нужных данных зачастую изменяет поведение параллельной программы и ее временные характеристики и порождает проблему *зонда*.

Отладочные среды должны включать в себя средства поддержки таких операций, как трассировка программы, установка контрольных точек, а также вывода и изменения состояния переменных, регистров и массивов данных.

В литературе описано несколько подходов к параллельной отладке. Среди них отладка *на уровне представления исходного текста* и отладка *на процессном уровне*.

Отладка на уровне представления исходного текста — традиционное исследование с текстом в руках состояния потока управления программы и ее данных. При этом возможен неприятный эффект наведенных ошибок (эффект домино), при котором ошибка, найденная поначалу, — вовсе не ошибка, а ее проявление в данном месте, тогда как ошибку надо искать совсем в другом.

При использовании средств отладки на процессном уровне процессы, как правило, представляют, абстрагируясь от их внутренней структуры, черными ящиками, взаимодействующими друг с другом. Временной аспект взаимодействия передается средствами анимации. За счет этого легко увидеть первый процесс, который генерирует ошибочные данные или ошибочно взаимодействует с другими, что и должно разрешить проблему наведенных эффектов. Функционально связанные между собой процессы можно объединять в виртуальные процессы и тем самым обеспечивать иерархическое рассмотрение структуры прикладной программы. На каждом иерархическом уровне представляется ограниченное число процессов, которые достаточно легко изучить. Наличие механизма зумминга помогает переходить на различные иерархические уровни.

Можно выделить два подхода к созданию средств отладки процессного уровня. При одном подходе используется анимация стандартных видов представлений. При другом отладчики отражают фактическую структуру алгоритмов. (При этом в качестве преимущества первого подхода отмечается, что не надо придумывать графические представления для конкретных приложений — каков бы ни был алгоритм, для него используется все та же картинка... Однако из-за общности образов не отражается реальная структура программы.) Отладчики, использующие графические образы, связанные с конкретной задачей, лучше вписываются в визуальные среды программирования. Так как иерархические визуальные образы создаются самим программистом в ходе проектирования, они лучше отражают имеющуюся у него ментальную картину программы.

Понятие *сравнительной отладки* определено через отличия с традиционными методами по двум основным вопросам. Во-первых, переменные отлаживаемой программы сравниваются не с представлениями пользователей о них, а с переменными, сгенерированными в контрольном варианте программы, о котором известно, что он выполнен правильно. Во-вторых, то, что контрольный вариант вырабатывает правильные значения, позволяет автоматизировать про-

цесс сравнения и выполнять его следующим образом. Сначала пользователь формулирует набор утверждений о ключевых структурах данных в контрольном и отлаживаемом вариантах программы. Эти утверждения описывают те позиции в обоих вариантах, в которых значения переменных должны совпадать, а различия в них указывают на ошибку. Далее уже отладчик отвечает за управление выполнением вариантов программы, проверяя выполнение условий путем сравнения структур данных и сообщая о различиях. Если появилось такое сообщение, пользователь пытается уточнить местоположение ошибки в программе, вторично вводя уточненные утверждения и вновь запуская отладчик. А когда местоположение ошибки в достаточной мере установлено, используются традиционные методы отладки.

Развитие отладочных визуальных систем началось в 80-е годы, когда были созданы такие системы, как Voyeur [24], Belveder [22], отладчик для вычислителя Victor [25] и др.

В отладчике Voyeur используются средства создания образов для представления параллельных программ, не зависящие ни от языка, ни от вычислительной системы. Характерным для этого отладчика является пример отладки программы, моделирующей поведение хищных и нехищных рыб. "Мир", в котором они живут, представляет собой сетку, состоящую из квадратов, где находятся рыбы, поведение которых рассчитывается на различных процессорах вычислительной системы с MIMD-архитектурой. Один из 16 процессоров управляет частью сетки из 4×4 квадратов. При отладке применялось естественное представление модельных объектов. Использование визуального отладчика позволило одновременно увидеть несколько участков мира обитания рыб и составить представление о ходе работы программы сразу на нескольких процессорах.

Визуальный отладчик параллельных систем Belveder использует образы, представляющие логическую схему межпроцессных связей. Анимация осуществляется за счет закраски квадратов, представляющих активные процессы, и рисования жирных линий со стрелками при использовании соответствующего канала. Даже такая простая анимация дает возможность выявить ошибки программы. Для различных задач используются различные процессорные структуры. Задача коммивояжера решается на процессорной структуре *гиперкуб*, в результате получается трехмерный образ коммуникационных событий. Задача динамического программирования решается на треугольном массиве процессоров. Решение графовых задач проводится на процессорных структурах типа *дерево*. Несмотря на использование средств увеличения наглядности, при интерпретации получающихся анимационных представлений, по мнению авторов, могут возникнуть определенные трудности, связанные со сложностью межпроцессных связей.

В системе отладки транспьютерной системы Victor при визуализации параллельного исполнения строится граф событий программы, включающий узлы таких типов, как начало, конец, ветвление и объединение процессов, их синхронизация, прием и посылка межпроцессорных сообщений в порядке того, как они произошли. Создан механизм стягивания графа, служащий для обеспечения сжатия информации при получении "крупнозернистой" визуализации параллельной программы.

В 90-е годы появилось новое поколение систем, среди которых следует отметить отладчики Panorama [26], Guard [27], p2d2 [28].

Panorama предлагает три встроенных графических вида и один текстовый вид отображений, позволяющих изучить состояние программы.

Графические виды отображений суть следующие:

1) *отображение процессоров* — показывает очереди сообщений, ожидающих чтения на каждом процессоре;
2) *временная линия* — показывает события взаимодействия с каждым процессором в течение некоторого *отрезка времени;*
3) *отображение массивов* — показывает в графической форме содержимое двумерного массива, распределенного между несколькими процессорами.

Вид *отображение процессоров* представляет собой граф, где узлами представлены процессоры, а окружающие их элементы служат для представления очередей непрочитанных сообщений. Линии, соединяющие процессоры, показывают логические связи между ними. Элементы, лежащие на линиях связи, представляют сообщения от соседнего узла (процессора) и имеют их номера в очереди. Внизу и справа от каждого узла находится еще один элемент, содержа-

щий счетчик сообщений, прибывших от несоседних узлов. Пользователь может редактировать топологию размещения процессоров.

Вид отображения *временная линия* показывает появление событий приема и передачи на отдельном процессоре.

Вид *отображение массивов* показывает содержание распределенного между несколькими процессорами двумерного массива, ставя в соответствие значениям его элементов различную интенсивность серого цвета. Этот вид отображения выглядит как матрица квадратных ячеек. Более светлые ячейки представляют бóльшие числа. Значения выше и ниже указанного пользователем предела показываются красным и синим цветом соответственно. Линии, пересекающие крест-накрест вид отображения, показывают распределение массива по процессорам. Указанием на ячейку, представляющую элемент массива, определяется его значение.

Текстовый вид отображения обеспечивает прямой интерфейс с базовым отладчиком в виде обычной командной строки. Пользователь может отлаживать программу, используя такие возможности, как установка контрольных точек различного назначения и проверка содержимого переменных.

Окно текстового интерфейса с базовым отладчиком дает возможность взаимодействовать с отладчиком, работающим на одном из процессоров параллельной ЭВМ. Доступ к текстовому интерфейсу пользователь всегда может получить через основное окно отладчика Panorama, а также указав на отдельный узел вида *отображение процессоров.*

Программист может обнаружить общую природу ошибки, используя визуальные методы, а потом перейти в окно базового отладчика и точно определить причину и местонахождение ошибки, применяя его средства.

Инструментарий отладчика Guard расширяет традиционные подходы к отладке путем облегчения сравнения структур данных двух работающих программ. Использование этой методики делает возможным применение ранних версий программы (о которых заведомо известно, что они функционируют правильно) для генерации значений с целью сравнения с новой разрабатываемой программой. Guard позволяет запускать контрольный и тестируемый варианты программы на различных ЭВМ. Простые методы визуализации дают пользователю возможность изучать различия в структурах данных, а использование потока данных — быстро выявлять ошибочные разделы программы.

Guard поддерживает три вида сообщений пользователю о наличии различий в версиях программы:

1) текстовый;
2) на базе использования битовых карт;
3) на базе использования трехмерных методик визуализации.

При текстовой выдаче печатаются фактические значения исследуемых переменных и различия между ними.

Использование битовых карт наиболее подходит при изучении массивов. В этом случае выдаются только два значения — максимального различия между соответствующими элементами массива и полной суммы различий между всеми элементами. Остальная полезная информация представляется в виде прямоугольной битовой карты на экране дисплея, на которой белыми пикселями обозначаются элементы массива, одинаковые в обоих вариантах программы, а черными — элементы с различными значениями. Такая, достаточно простая, визуализация массивов особенно полезна при определении ошибок, связанных с заданием циклов или адресных выражений, так как эти типы ошибок обычно генерируют четко различаемые образы.

За счет применения средств научной визуализации, обеспечивающих вывод трехмерных объектов, поддерживаются наиболее эффективные методики визуализации различий, которые лучше всего подходят для анализа многомерных массивов. Возможно применение анимации для отображения развития в различиях, возникающих по ходу выполнения программы. Основой вида отображения служит вывод трехмерного ограниченного пространства с нанесенной по осям разметкой. Трехмерные цветные поверхности отражают различия, полученные при расчете разных вариантов реализации, например, моделей атмосферных явлений. Авторы называют такой вид отображения *поверхностью ошибок.*

По мнению авторов p2d2, в отладчиках существует две наиболее серьезные проблемы с пользовательским интерфейсом, разрабатываемым для многопроцессных программ. Первая проблема заключается в создании механизма указания того, какие из процессов программы должны получить (и затем выполнить) соответствующие операции управления. Вторая проблема связа-

на с необходимостью обеспечивать пользователя методами поиска и извлечения информации о состоянии нужного процесса из сложного состояния всей многопроцессной программы. Так как решение обеих проблем требует описания интересующих пользователя процессов по ходу отладки, авторы p2d2 называют данную стратегию использования отладчика *отладочной парадигмой процессной навигации.* В отладчике p2d2 процессную навигацию поддерживают три вида отображения:

1) *процессная сетка* — представляет все процессы, участвующие в вычислениях;
2) *группа процессов* — содержит информацию о нескольких процессах;
3) *процесс* — детализирует данные об одном процессе.

Возможно создание дополнительных видов отображения за счет подключения средств вывода структур данных, заимствованных из стандартных систем визуализации.

Во второй половине 90-х годов для отладки программ в рамках парадигмы передачи сообщений была разработана интересная визуальная система MAD [29], активно использующая представление *графа событий.* (Интересно, что так же назывался и другой визуальный отладчик параллельных программ, разработанный в 80-е годы совершенно другой командой исследователей и тоже использовавший граф событий [30].)

Граф событий визуально представлен в виде диаграмм пространства-времени. События, произошедшие при выполнении программы, размещаются в соответствии со временем и номером процесса (номера процессов задаются на вертикальной оси, а время — на горизонтальной). События взаимодействия (прием и посылка) показываются при помощи стандартных стрелок. Существуют также события чтения/записи как для отдельных переменных, так и для распределенных массивов. Использование графа событий позволяет выводить ряд характеристик исследуемой программы автоматически.

В параллельных программах установка контрольных точек является нетривиальной задачей, так как необходимо получить согласованный срез состояний нескольких взаимодействующих процессов. При текстовом отображении эта отладочная задача практически невыполнима. В отладчике MAD расстановка контрольных точек осуществляется графически. Сначала пользователь на диаграмме пространства-времени выбирает одно событие, при котором выполнение процесса будет приостановлено. Затем отладочный инструментарий генерирует срез состояний путем определения других контрольных точек на всех процессах, которые подвергаются влиянию выбранного пользователем контрольного события.

Одной из задач отладки является выбор изучаемого объекта. Как правило, он осуществляется путем указания имени объекта и/или его местоположения в памяти. Вывод состояния объекта, в частности распределенных массивов, обеспечивается отладчиком MAD за счет использования развитого набора трехмерных видов отображения, заимствованных из сред научной визуализации.

Отладчик MAD также предназначен для поддержки анализа и отладки недетерминизма параллельных программ, для чего необходимо выделить в программе события недетерминизма, связанные, например, с генерацией случайных чисел или гонкой сообщений. В отладчике разработаны виды отображения на базе графа событий, а также средства, позволяющие осуществлять различные манипуляции с событиями, включая перезапуск программы с другими параметрами для изучения гонки сообщений.

Выделим важную тенденцию в развитии систем визуальной отладки параллельных программ. Если в 80-е годы все эти системы базировались на посмертной визуализации собранных данных о работе параллельной программы, то в последние годы реализуются интерактивные визуальные отладчики параллельных программ. Так, в середине 90-х годов были созданы механизмы слежения за параллельно работающими процессами, в частности на базе алгоритма сортировки графа событий [31], который позволяет организовать непрерывное упорядочение и компоновку поступающей информации о событиях. И этот алгоритм, и другие подходы, реализованные, например, в отладчике Panorama, служат основой живой визуализации, весьма необходимой для отладки параллельных вычислений.

Следует отметить, что, несмотря на интересные решения, лежащие в рамках инженерии программного обеспечения (Software Engineering), проблема отладки правильности параллельных вычислений не решена. Одна из причин, по мне-

нию авторов, заключается в том, что не выбраны основные сущности отладки и, как следствие, разработанные виды отображения не позволяют в полной мере решить задачи представления отладочных данных для масс-параллельных вычислений.

Опыт авторской разработки среды визуализации для системы параллельного программирования DVM пока не позволяет сделать какие-либо заключения. Дело в том, что отладка параллельной правильности в DVM проводится на однопроцессорном (последовательном) варианте программы и не требует новых методов параллельной отладки. Правда, в DVM предусмотрен такой способ параллельной отладки правильности, как трассировка. Под трассой понимается последовательность действий (чтение/запись), предпринимаемых программой в отношении ее распределенных данных. Сбор трассы производится при специальном режиме компиляции и запуска отлаживаемой программы.

По заказу разработчиков DVM реализованы средства динамического вывода трассы программы, позволяющие:

- работать с трассой, группируя и сворачивая в блоки трассу, относящуюся к циклу или фрагменту задачи;
- имитировать динамику выполнения программы путем подсвечивания строк трассы и соответствующих строк исходного текста.

Работа выполнена при поддержке Российского фонда фундаментальных исследований (коды проектов 01-07-90210, 01-07-90215).

Список литературы

1. *Browne S., Dongarra J., London K.* Review of Performance Analysis Tools for MPI Parallel Programs. http://www.cs.utk.edu/~browne/perftools-review/.
2. *Stankovich N., Zhang K.* Visual programming for message-passing systems // International Journal of Software Engineering and Knowledge Engineering. 1999. Vol. 9, No 4. P. 397—423.
3. *Zhang K., Hintz T., Ma X.* The role of graphics in parallel program developing // Journal of Visual Languages and Computing. 1999. No 10. P. 215—243.
4. *Kranzlmuller D. Schaubschlager Ch., Volkert J.* Why debugging parallel programs needs visualization // Proc. VMPDP 2000, Workshop on Visual Methods for Parallel and Distributed Programming 2000. Seattle, WA, USA. September 2000.
5. *Авербух В. Л.* Средства визуализации параллельного программирования (обзор) // Пользовательский Интерфейс: исследования, проектирование, реализация. 1993. № 4. С. 32—41.
6. *Авербух В. Л.* Визуальная отладка параллельных программ (обзор) // Алгоритмы и программные средства параллельных вычислений. 1995. С. 21—46.
7. *Авербух В. Л., Байдалин А. Ю.* Проектирование средств визуализации для системы параллельного программирования DVM // Там же. 2001. Вып. 5. С. 3—40.
8. *Авербух В. Л., Байдалин А. Ю.* Проектирование визуальных средств разработки программ для системы параллельного программирования DVM // Там же. 2002. Вып. 6. С. 3—33.
9. *Коновалов Н. А., Крюков В. А.* Параллельные программы для вычислительных кластеров и сетей // Открытые системы. 2002. № 3. С. 12—18.
10. *Pong M. C.* A Graphical language for concurrent programming // Proc. of the 1986 IEEE Workshop on Visual Languages. June 1986. P. 26—33.
11. *Kacsuk P., Dozsa G., Fadgyas T.* Designing parallel programs by the graphical language GRAPNEL // Microprocess. and Microprogramm. 1996. Vol. 41, No 8—9. P. 625—643.
12. *Al-Mulhem M., Ali Sh.* Visual Occam: syntax and semantics // Comput. Lang. (Gr. Brit.) 1997. Vol. 23, No 1. P. 1—24.
13. *Beguelin A. L., Nutt G. J.* Visual parallel programming and determinnacy: a language specification, an analysis technique, and a programming tool // Journal of Parallel and Distributed Computing. 1994. Vol. 22(2). P. 235—250.
14. *Stankovic N, Kranzlmuller D., Zhang K.* The PCG: an empirical study // Journal of Visual Languages and Computing. 2001. Vol. 12, No 2. P. 203—216.

15. *Philippi S.* Visual programming of concurrent object-oriented systems // Journal of Visual Languages and Computing. 2001. Vol. 12, No 2. P. 127—143.
16. *Webber J., Lee P. A.* Visual, Object-Oriented Development of Parallel Applications. www.math.uni–muenster.de/cs/u/guidow/CONF/wsvl2000/Papers/webbervl2000.pdf.
17. *Stankovic N., Zhang K.* Towards visual development of message-passing programs // Proc. 1997 IEEE Symposium on Visual Languages. Isle of Capri, Italy. Los Alamitos, Ca. September 23—26, 1997. P. 144—151.
18. *Stone J. M.* A graphical representation of concurrent processes // Proc. of the ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging. University of Wisconsin, Madison, Wisconsin. May 5—6, 1988. SIGPLAN Notices. 1989. Vol. 24, No 1. P. 226—235.
19. *Важенин А. П. Миренков Н. Н.* Элементы системы визуального программирования // Программирование. 2001. № 4. С. 68—80.
20. *Вшивков В. А., Краева М. А., Малышкин В. Э.* Параллельная реализация метода частиц // Там же. 1997. № 2. С. 39—51.
21. *Yu Y., D'Hollander E. H.* Loop parallelization using the 3D iteration space visualizer // Journal of Visual Languages and Computing. 2001. Vol. 12, No 2. P. 163—181.
22. *Hough A. A., Cuny J. E.* Initial experience with a pattern-oriented parallel debugger // Proc. of the ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging. University of Wisconsin, Madison, Wisconsin. May 5—6, 1988. SIGPLAN Notices. 1989. Vol. 24, No 1. P. 195—205.
23. *Васев П. А., Перевалов Д. С.* О создании методов многомерной визуализации // Тр. XII Межд. конф. по компьютерной графике и машинному зрению "Графикон 2002". Нижний Новгород: ННГУ, 2002. С. 431—437.
24. *Socha D., Bailey M. L., Notkin D.* Voyeur: graphical views of parallel programs // Proc. of the ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging. University of Wisconsin, Madison, Wisconsin. May 5—6, 1988. SIGPLAN Notices. Vol. 24, No 1. 1989. P. 206—215.
25. *Zernik D., Snir M., Malki D.* Using visualization tools to understand concurrency // IEEE Software. 1992. Vol. 9, No 3. P. 87—92.
26. *May J., Berman F.* Retargetability and extensibility in a parallel debugger // Proc. ACM/ONR Workshop on Parallel and Distributed Debugging. May 1993.
27. *Abramson D., Foster I., Michalakes J., Sosic R.* Relative debugger: a new methodology for debugging scientific applications // Communication of the ACM. Vol. 39, No 11. P. 69—77.
28. *Cheng D., Hood R.* A portable debugger for parallel and distributed programs // Proc. of Supercomputing'94. http://www.nas.nasa.gov/NAS/Tools/Projects/P2D2/.
29. *Kranzlmuller D., Schaubschlager Ch., Volkert J.* A brief overview of the MAD debugging activities // Proc. AADEBUG 2000, 4th Intl. Workshop on Automated Debugging. Munich, Germany. August 2000.
30. *Rubin R. V., Rudolph L., Zernik D.* Debugging parallel programs in parallel // Proc. of the ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging. University of Wisconsin, Madison, Wisconsin. May 5—6, 1988. SIGPLAN Notices. 1989. Vol. 24, No 1. P. 216—225.
31. *Kimelman D, Zernik D.* On-the-Fly Topological Sort-A Basis for Interactive Debugging and Live Visualization of Parallel Programs. http://swt-www.informatik.uni-hamburg.de/~1friedri/sv/references/On-the-Fly_Topological_Sort.ps.gz.