Состояние дел в визуализации программного обеспечения параллельных вычислений

В.Л. Авербух, А.Ю. Байдалин, Д.Р. Исмагилов, А.Ю. Казанцев
ИММ УрО РАН, Уральский Государственный Университет
Екатеринбург, Россия
averbukh @ imm.uran.ru
Работа выполнена при поддержке гранта РФФИ № 04-07-90120

Аннотация

Сделан обзор состояния дел в визуализации программного обеспечения параллельных вычислений, включая визуальные языки и визуальные отладчики правильности и производительности. Проведено обсуждение обзора и сделаны некоторые прогнозы развития.

Ключевые слова: *Визуализация программного обеспечения, параллельные вычисления.*

1. ВВЕДЕНИЕ

Под визуализацией программного обеспечения понимается совокупность методик использования графики и средств человеко-машинного взаимодействия, применяемых для лучшего уяснения понятий и эффективной эксплуатации программного обеспечения ЭВМ, а также для спецификации и представления программных объектов в процессе создания программ. В число систем визуализации программного обеспечения согласно ряду классификаций входят системы визуального программирования и системы визуализации программирования, а также системы программирования путем демонстраций в той части, где последние используют визуальные методы представления образцов вводимой и выводимой информации.

Визуализация программного обеспечения параллельных вычислений включает в себя исследования и разработки визуальных языков параллельного программирования, визуальных отладчиков и систем настройки, отладки и измерения производительности параллельных программ.

Первые системы визуализации для параллельного программирования появились еще в 80-х годах. Достаточно активно развивались как языки визуального параллельного программирования, так и визуальные отладчики правильности и системы настройки производительности параллельных и распределенных программ. На середину 90-х годов пришелся расцвет визуализации программного обеспечения параллельных вычислений. Это связано с проблемами, возникающими при создании программных комплексов для параллельных супервычислителей с огромным количеством процессоров (конечно, по тогдашним меркам). В ведущих академических центрах и лабораториях аэрокосмических и ядерных ведомств США были в разработаны мощные системы. Были получены интересные результаты и в других странах, включая Россию. Тогда казалось, что возможности визуализации параллельного программирования (также как и самого параллельного программирования) будут расти также быстро, как и мощности параллельных вычислителей, давая тем самым новый инструмент для программистов. Однако анализ показывает замедление темпов развития в последующие годы, отсутствие свежих идей и результатов, соответствующих современным супервычислителям. Частично это связано с определенным разочарованием в результатах развития всей визуализации программного обеспечения, частично с некоторыми кризисными явлениями в параллельных вычислениях, когда возможности программирования отстают от все возрастающих возможностей техники. Выход из этой ситуации видимо следует искать за счет использования принципиально новых подходов, как в области визуализации, так и самого параллельного программирования.

Состояние дел в визуализации программного обеспечения параллельных вычислений отслеживается последние двадцать лет. По этой проблеме опубликован целый ряд обзорных работ, в том числе и написанных нами в последние годы [1-6]. Поэтому здесь мы не будем подробно останавливаться на характеристиках каждой системы, а обобщим основные тенденции, выявляемые из опыта развития, рассмотрим возникающие проблемы и внесем некоторые предложения по их решению.

2. ВИЗУАЛЬНЫЕ ЯЗЫКИ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ

При разработке визуальных языков программирования в качестве основного подхода используется рисование графов, как правило, отображающих поток управления либо поток данных. При этом в графических нотациях используются диаграммы и схемы, зачастую заимствованные из «бумажных» технологий программирования, или специально придуманные для данного случая иконические знаковые системы.

В ранних визуальных языках при проектировании программы на графовых схемах могла отображаться потенциальная возможность параллельного выполнения кода. Обычно в этих языках параллельные дуги графа описывали потенциально параллельные участки программы. При этом реальное создание эффективного параллельного кода возлагалось не на программиста, а (и то больше в теории) на систему программирования.

На следующем этапе развития визуальных языков параллельного программирования графы используются уже для создания настоящей параллельной программы с описанием взаимодействия ее процессов. Во многих случаях такие языки базируются на графы потока данных. Например, этот граф может показывать создание данных одним последовательным вычислением для нужд другого. Параллельность, как и в ранних системах, задается

дополнительной (параллельной) дугой. При программировании вначале рисуется граф потоков данных решения данной задачи, а на следующем этапе за счет текстового описания портов ввода и вывода процесса аннотируются узлы графа (последовательные процессы), а также пишутся правила запуска этих узлов.

Существуют примеры систем, где программа на визуальном языке состоит из потока управления, потока данных и множества интерпретаций узлов (то есть процедур, запускаемых при достижении узла). Граф потока управления языка в этом случае обеспечивает представление последовательных потоков, переключение управления, параллелизм и синхронизацию. Параллелизм поддерживается в основном за счет описания того, как несколько маркеров, описывающих поток данных, могут распространяться по графу. На язык и систему программирования могут возлагаться задачи уже на этапе проектирования и «визуального кодирования» добиться правильности (в частности детерминированности) и эффективности параллельных программ.

В дальнейшем в развитии визуальных языков параллельного программирования возобладала тенденция четкого изображения взаимодействия параллельных процессов, построенного по правилам библиотек PVM или MPI. Например, рисуется картинка, состоящая из одной или более графовых схем, представляющих структуру процессов для параллельной программы. Процессы могут взаимодействовать друг с другом посредством однонаправленных связей через порты приема или передачи. В описаниях процессов кроме визуальной схемы программы имеются тексты-аннотации на традиционных языках программирования. По сути, это гибридные системы, где графика используется для того, чтобы изобразить те части программы, которые содержат важные элементы параллелизма, а текстовые описания применяется, например, для декларации и определения данных, сегментов программы, не содержащих обменов и т.д. Основная задача графических представлений - обеспечить высокоуровневые абстракции распределенных программ, в которых ключевым является обеспечение взаимодействия процессов. В развитых системах существует графическая поддержка структурированного проектирование 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.

Существует пример скрупулезной визуальной реализации практически всех возможностей языка параллельного программирования Occam. При общем графовом описании структуры программы для каждой программной конструкции используется та или иная абстрактная пиктограмма (икон). Графический синтаксис также базируется на языке Occam.

Тенденция изображать в визуальном виде распараллеливание процессов или их потенциальное распараллеливание может быть реализована не только за счет использования графов потока управления и потока данных. Так в одной из систем визуального параллельного программирования вводится понятие графа взаимодействия процессов. В свою очередь граф взаимодействия процессов опирается на понятия «карта параллельности» и «пространственно-временная диаграмма».

Эти методики отображения параллельности ранее использовались на этапах отладки и настройки эффективности параллельной программы. При реализации визуального языка параллельного программирования было проведено их расширение с тем, чтобы позволить их использование и на этапе конструирования программы.

Один из последних по времени примеров разработки языка визуального параллельного программирования интересен реализацией модели параллельного исполнения потока объектов. В этом случае объединяется объектно-ориентированную модель и модель потока данных. Таким образом, существует возможность обеспечить как задание 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.

Кроме превалирующих графовых языков визуального программирования параллельных вычислений следует отметить подход, близкий к идеям программирования путем демонстраций.

Предлагаемая система базируется на визуализации как параллелизма, так и динамики обработки данных, а также на прямом отображении таких визуальных спецификаций непосредственно в программу конкретной ЭВМ. При этом считается, что уже существует кем-то заранее созданный набор вычислительных схем, представленных библиотекой цветных фигур, картинок и анимационных фильмов с звуковым сопровождением. Эти фильмы являются серией кадров (вычислительных шагов). Пользователь может преобразовать серии кадров и подмножества узлов и объектов. Далее его задача состоит в описании вычислений на подмножествах или на их элементах. При демонстрации кадров различных уровней иерархии применяются как цвет, так и звуковое сопровождение. Визуальные образы должны представлять высокоуровневые математические объекты, например, параметризированную матрицу и вектор при описании методов решения задач линейной алгебры.

К визуальным системам параллельного программирования можно отнести пакеты, где непосредственные манипуляции с графическими объектами позволяют задать данные, необходимые для работы прикладной параллельной программы. В конкретном случае реализация счетных алгоритмов связана с использованием прямоугольной сеткой. В определенных методом точках сетки вычисляются сеточные функции, описывающие, в частности, действующие в ней силы. Программирование заключается в задании вычислительных функций внутри каждой ячейки, которые определяют и состояние соседних ячеек. Пользователь может отслеживать ход программирования, наблюдая собранное из «кирпичей»-ячеек пространство моделирования вместе с заданными вычислениями. Распараллеливание осуществляется компилятором автоматически.

Таким образом, мы видим, что методы представления для визуальных языков параллельного программирования разбиваются на два класса - графовые и «квазиестественные» (то есть естественные для заданной прикладной области). средствами параллельных вычислений.

3. ВИЗУАЛЬНАЯ ОТЛАДКА ПРАВИЛЬНОСТИ

Теперь перейдем к рассмотрению ситуации с визуальной отладкой правильности параллельных программ. Развитие таких отладочных визуальных систем началось в 80-х годах.

В параллельных программах возникают новые по сравнению с последовательными программами классы ошибок, соответственно которым должны разрабатываться и новые методики отладки. Естественно для отображения сложной динамики взаимодействия параллельных процессов прикладной программы использовать динамическую визуализацию. Кроме этого необходимо использование базовых текстовых отладчиков, работающих на отдельных процессорах вычислительного комплекса.

Выделим основные задачи, возникающие при разработке систем визуальной отладки параллельных программ.

Прежде всего, это задача сбора информации. Существует два основных подхода к сбору информации - использование контрольных точек и регистрация событий, происходящих в процессах (как правило, событий ввода-вывода). И в том и в другом случае возникают проблемы с изменением временных характеристик параллельных программ. Как системы, основанные на контрольных точках, так и системы, основанные на регистрации событий, стараются минимизировать свое влияние на время выполнения программы, но, конечно, полностью избежать этого не могут.

Вторая задача связана с организацией работы с отладочными данными. Реализуются системы, работающие в режиме непосредственного «живого» (on-line) взаимодействия или на базе «посмертной» (off-line) обработки собранных данных. В связи уже упомянутыми проблемами изменения временных характеристик параллельных программ при любом вмешательстве в их работу ясно, что организация отладки в on-line режиме возможно только при использовании сложных решений в области инженерии программного обеспечения (Software Engineering). Первоначально эти проблемы решались за счет моделирования выполнения параллельных процессов на последовательных компьютерах. Позднее в ходе реализации различных отладочных систем были разработаны изощренные механизмы для фиксации временных характеристик параллельных программ и компенсации вторжения в их функционирование. Отметим, что проблема on-line визуализации также актуальна и для систем отладки и настройки производительности параллельных программ.

Третья задача - это задача выбора методик визуальной отладки и поиск соответствующих видов отображения данных. По этому поводу следует отметить чрезвычайную бедность репертуара видов отображения, что отражает недостаточную проработку проблемы отладки параллельных программ. Имеет место явное отставание методологии отладки (включая и методологию визуализации) от интенсивно развивающихся аппаратных и программных параллельных технологий. Необходим анализ видов отображения в визуальных параллельных отладчиках, что в дальнейшем поможет выявить основные идеи отладки и помочь в конструировании новых метафор визуализации.

Как и визуальные языки параллельного программирования, визуальные отладчики параллельных программ появились в 80-х годах. Сразу же выделились два класса методик отладки - сравнительно традиционная отладка на уровне текста программы и отладка на процессном уровне.

При использовании средств отладки процессного уровне процессы, как правило, представляют, абстрагируясь от их внутренней структуры, черными ящиками, взаимодействующими друг с другом. Временной аспект взаимодействия обычно передается за счет анимации.

Существуют несколько подходов к созданию видов отображения при отладке на процессном уровне. Один из них предусматривает анимацию параллельных алгоритмов и программ с использованием естественной для данного приложения образности. В другом используется анимационое представление взаимодействия процессов, основанное на образах, представляющих логическую схему межпроцессных связей. За счет этой методики относительно легко обнаружить первый процесс, который генерирует ошибочные данные или ошибочно взаимодействует с другими, что и должно разрешить проблему наведенных эффектов. Также в основу отладки могут быть положены некоторые абстракции (например, граф событий параллельной программы или карта параллельности), которые и служат для создания соответствующих видов представлений.

В отладчиках, использующих связанные с конкретной задачей графические образы, в одних случаях могут применяться стилизованные образы реальных объектов, связанных с физической моделью, как в примере параллельной реализации модели хищник-жертва, где при визуализации перемещались образы акул и их жертв-рыб. В других случаях хороший результат дает использование традиционного отображения математических объектов - двумерных и трехмерных графиков функций, изолиний и изоповерхностей, векторных полей и пр. Образы, связанные с конкретной задачей, наглядны и легки для интерпретации пользователем. Тем более, если визуальные образы создаются самим программистом в ходе проектирования (а такая возможность существовала почти во всех отладчиках). Тогда эти образы лучше отражают имеющуюся у пользователя ментальную картину программы. Наконец, такие отладчики хорошо вписываются в визуальные среды программирования. Однако данный подход в принципе является глубоко специализированным и требует в каждом новом случае переделки и образности, и интерфейса соответствующих графических объектов с отлаживаемыми программами. Всякий раз надо не только придумывать картинки, но и создавать средства для их реализации. Обычно для этого в отладчиках предусмотрен пользовательский инструментарий для проектирования образов и их связывания с программными объектами. Но в сущности все это просто перекладывает задачу проектирования видов отображения на самого пользователя. Вероятно все эти факторы привели к тому, что в современных отладочных системах значительно реже используются образы, связанные с физическими или математическими моделями, лежащими в основе отлаживаемой параллельной программы.

Представления логической структуры межпроцессных (межпроцессорных) связей вместе с простыми приемами, создающими анимационный эффект (например, за счет закраски квадратов, представляющих активные процессы, и рисования жирных линий со стрелками для обозначения активных каналов) дают возможность выявить ошибки программы определенного класса. Функционально связанные между собой процессы можно объединять в виртуальные процессы и тем самым обеспечивать иерархическое рассмотрение структуры прикладной программы. Тогда на каждом иерархическом уровне представляется ограниченное

число процессов, которые достаточно легко изучить. Механизм зуминга может помочь переходить на различные иерархические уровни. Однако, несмотря на использования средств увеличения наглядности, при интерпретации получающихся анимационных представлений большого числа процессов возникают трудности, связанные со сложностью межпроцессорных связей.

Недостатки данного подхода, возможно, связаны также с тем, что не все ошибки параллельных программ сводятся к ошибкам синхронизации передачи данных. Поэтому необходимы другие методики представления как параллельных процессов, так и их «интересных событий», в число которых могут входить не только обмены сообщениями, но и пользовательские контрольные точки в программе, системные события (например, события динамического порождения и уничтожения процессов) или аппаратные события.

В этом случае могут вводиться новые абстракции, например, граф событий параллельной программы, включающий узлы, представляющие такие события, как начало, конец, ветвление и объединение процессов, их синхронизация, приём и посылка межпроцессорных сообщений. Предусмотрен механизм стягивания графа, служащий для обеспечения сжатия информации, для получения «крупнозернистой» визуализации параллельной программы. Вывод графа событий, отображающего выполнение параллельной программы, может помочь в понимании поведения системы, состоящей из большого числа процессоров.

Другой метод графического представления параллельных процессов связан с уже упоминавшейся «картой параллельности».

Также популярен в целом ряде отладчиков вид отображения «временная линия», который показывает появление событий приема и передачи на отдельном процессоре.

В отладчиках 90-х годов, как правило, используются несколько типов видов отображения - и отображение структуры межпроцессного взаимодействия, и отображение данных, и текстовый интерфейс с базовым отладчиком дает возможность взаимодействовать с отладчиком, работающем на одном из процессоров параллельного вычислителя.

В параллельных отладчиках появляются задачи описания интересующих пользователя процессов по ходу отладки, которые можно решить за счет «процессной навигации», которая может поддерживаться за счет перехода от одного вида отображения к другому, например, от вида, представляющего все процессы, участвующие в вычислениях к виду «группа процессов», содержащего информацию о нескольких процессах, и, наконец, к виду «процесс», в котором детализированы данные об одном процессе.

Как уже отмечалось, в целом репертуар видов отображения очень беден, что связано с ограниченным числом методик отладки параллельных программ. В этой связи особенно интересна идея «сравнительной отладки», которая 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, служащих для сравнения с новой разрабатываемой программой. Необходим запуск контрольного и тестируемого вариантов программы на различных ЭВМ. Достаточно простые методы визуализации (текстовые сообщения, битовые карты, поверхности ошибок) позволяют пользователю изучать различия в структурах данных, а использование потока данных дает возможность быстро выявить ошибочные разделы программы.

Использование битовых карт наиболее подходит при изучении массивов. В этом случае выдаются только две величины - максимальное различие между соответствующими элементами массива и полную сумму разностей между всеми элементами. Остальная полезная информация выдается в виде прямоугольной битовой карты на экране дисплея, на которой белыми пикселями обозначаются значения массива, одинаковые в обоих вариантах программы, а черными - элементы с различными значениями. Существуют примеры использования для подобных целей (указания ошибок на массивах) цветных битовых к.арт.

Трехмерные цветные поверхности («поверхности ошибок») могут отражать различия, полученные при расчете различных вариантов реализации некоторой модели. Возможно применение анимации для отображения развития в различиях, возникающих по ходу выполнения программы.

Итак, при разработке видов отображения в визуальных отладчиках существует два подхода – использование предопределенных видов отображения и возможность создания пользователем собственного набора представлений. В первом случае пользователь должен освоить некоторый (возможно новый для себя) тип образности. Во втором (как мы уже отмечали) - разработчики по сути перекладывают на пользователя задачу проектирования видов отображения. Разработка видов отображения остается существенной проблемой визуализации программного обеспечения параллельных вычислений.

Отмеченное противоречие между достижениями в области инженерии программного обеспечения и ограниченным числом видов отображения проявляется также в системах отладки и настройки производительности параллельных программ.

4. ВИЗУАЛЬНАЯ ОТЛАДКА И НАСТРОЙКА ПРОИЗВОДИТЕЛЬНОСТИ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

Очевидно, что получение максимально эффективных параллельных программ является основной задачей проектирования. Однако на практике случается, что с большими усилиями распараллеленная программа работает чуть ли не медленнее, чем ее последовательный вариант. Есть примеры того, как мощные программистские силы бросаются на достижение сравнительно небольшого ускорения программного комплекса, что, однако, дает огромный эффект, так как позволяет вписаться в суточный цикл промышленного счета. Все это показывает, что необходимы значительные усилия для создания средств отладки и настройки производительности (эффективности) параллельных программ.

Системы отладки эффективности служат для того, чтобы предсказывать, находить и избегать возможную неэффективность исполнения параллельных программ. Визуальный вывод данных об эффективности использования

параллельных систем должен быть прямо связан с заданной моделью производительности. Визуализация должна проектироваться и применяться в рамках интегрированной среды оценки производительности. Эффективное распараллеливание требует, среди всего прочего, чтобы вычислительные работы были равномерно распределены между процессорами, чтобы каждый процессор выполнял свою долю работы одновременно с другими, чтобы были минимизированы накладные расходы на работу данного последовательного алгоритма и т.п. Таким образом, соответствующие сущности, служащие для анализа эффективности и производительности работы процессоров (итоговая нагрузка каждого процессора, распараллеливание их работы и накладные расходы), важны для оптимизации производительности. Для количественной оценки эти сущности должны отображаться в поддающиеся измерению понятия, описывающие эффективность параллельных вычислений, такие как время занятости процессоров, чрезмерные накладные расходы и время, затраченное на взаимодействие процессов. Именно для этих величин возможен мониторинг, то есть по ходу вычислений они могут отслеживаться и информация о них записываться в специальные файлы. В свою очередь сущности, связанные с анализом производительности, должны также отображаться на подходящие визуальные абстракции, например, такие как столбчатые диаграммы или гистограммы. Наконец, на последнем шаге (возможно после ряда промежуточных действий) данные, описывающие эффективность параллельных вычислений, отображаются в определенные виды отображений для графического вывода. Как правило, не удается в одном виде отображения показать все данные, необходимые для полноценного анализа производительности.

Большинство систем отладки эффективности, разработанных в конце 80-х - начале 90-х гг., используют отображение метрик производительности. Метрика - зависящая от времени функция, характеризующая некоторые аспекты производительности параллельной программы, например, нагрузка центрального процессора, использование памяти, число операций с плавающей запятой. Именно отображением метрик работы параллельных программ и занимаются системы отладки эффективности. Для визуального представления метрик часто применяются виды отображения, заимствованные из статистической графики.

Неофициальным образцом системы отладки эффективности к середине 90-ых годов стала система ParaGraph, которая поддерживает более 25 заимствованных из статистической графики видов отображения, среди которых, как хорошо известные столбчатые и круговые диаграммы и графики различных типов, так и менее привычные, например, диаграммы Гантта (отображения, при котором каждому параллельному процессу соответствует своя горизонталь, и разными цветами показывается время ожидания, простоя и работы данного процесса), или диаграммы Кивиата, несколько напоминающие круговые диаграммы и используемые для динамического представления занятости каждого процессора и баланса распределенных между ними вычислений. Многие системы отладки эффективности содержат примерно тот же набор визуальных средств, который представлен в системе Paragraph - диаграммы Гантта, графики и гистограммы различных типов. Созданы трехмерные аналоги диаграмм Гантта - цветные цилиндры, отображающие состояния параллельных процессов, связанные между собой линиями, представляющими обмены сообщениями.

В системах отладки производительности следующего поколения для проектирования системы видов отображения активно используются абстракции, например, такие, как модель поиска по осям «ПОЧЕМУ», «ГДЕ», «КОГДА». (Почему данная прикладная программа имеет низкую производительность? Где расположено «узкое место» производительности? Когда проявилось воздействие данного «узкого места» на производительность программы?) «Узкие места» производительности определяются путем автоматического поиска в информационном пространстве, определенном этой моделью.

Кроме отладки эффективности на уровне операций посылки сообщений можно проводить оптимизацию самого кода программы при ее распараллеливании. Существуют примеры систем, где поддерживается визуализация и последующие манипуляции с набором параметров параллельной программы, каждый из которых отражает различные аспекты производительности. Изменение этих параметров пользователем обеспечивает выбор оптимальных стратегий программирования, связанных с распараллеливанием. Анализируя значения параметров, пользователь имеет возможность выбрать стратегию распределения по процессорам структур данных и распараллеливаемых программных конструкций.

Характерны проявившиеся в середине 90-х годов противоречия между развитием технологий (как инженерии программного обеспечения, так и компьютерной графики) с одной стороны, и отставанием когнитивного аспекта визуализации параллельных вычислений - с другой. Например, система настройки и предсказания производительности параллельных программ была снабжена мощными средствами исследования исходного текста прикладной программы и вставки в нужные точки инструментальные средства, а также средствами компенсации этих вторжений в программу. Компенсатор внедрения исправляет файл трассировки, который может затем служить входным файлом для разнообразных средств оценки производительности, определяющих характеристики производительности на основании «посмертных» данных. При этом инструментатор работал с различными программными реализациями парадигмы передачи сообщений. В состав системы входило еще несколько интересно реализованных подсистем, служащих для поддержки различных аспектов процесса отладки и настройки производительности. В тоже время возможности визуализации в этой системе были достаточно ограничены. Анимационные виды отображения, которые могут реализоваться, например, за счет обновления предыдущего состояния, представляют информацию, указывая, когда выполняются отдельные программные конструкции, когда посылаются сообщения, как долго сообщения стоят в очереди перед тем, как обработаться, когда отдельный процессор простаивает. Возможно отображение фрагментов истории программы за счет непрерывной прокрутки изображения.

Аналогично в другой системе для вывода данных о производительности использовались мощные средства театра виртуальной реальности - куба размером с комнату, где на стены, пол и потолок могут проецироваться видеоизображение с высокой разрешающей способностью. При проектировании системы использовались интересные

идеи по представлению понятий оценки производительности (например, траектории поведения процессоров, решающих задачу в составе параллельного вычислителя) и метафоры визуализации. Основной метафоры визуализации являлась «матрица трехмерных выводов» (scattercube). При работе с системой пользователь как бы оказывается внутри трехмерного помещения, в котором «полу» и двух «стенах» выводятся кривые, описывающие поведение параллельной программы, то есть метрик производительности. Множество таких помещений, возникающее за счет возможности виртуальных перемещений в информационном пространстве оценок, по соответствующей траектории составляет виртуальный «небоскреб», путешествие по которому дает возможность полного исследования данных о производительности прикладной программы. Все это производит большое впечатление, но анализ сотен таких метрик представляется делом весьма непростым. Реализованная возможность «облета» помещений с целью изучения производительности большой последовательности процессов также может привести к сомнительным результатам из-за особенностей физиологии человека.

Существуют примеры использования интересных метафор для создания отображения элементов и состояний параллельной ЭВМ на тела и силы некоторой динамической системы. Моделирование параллельной ЭВМ ведется на основе топологии ее физической сети. Так для параллельного вычислителя вычислительные модули, вычисления, загруженные на данном модуле, коммуникационная сеть и количество обменов могут отображаться на тела, массу тела, пружины между телами и силы притяжения между телами соответственно. Затем можно выписать уравнения движения, которые моделируют данную динамическую систему. Далее движение системы моделируется посредством численного решения дифференциальных уравнений движения. Результат решения может представляться с использованием стандартных методик, хорошо известных в научной визуализации.

В этом случае требуется двойная интерпретация результатов - сначала результатов визуализации решения дифференциального уравнения, а потом отображенного на него состояния параллельного вычислителя.

Еще один подход к настройке производительности параллельных систем связан с настройкой производительности единичного процессора и решения таких проблем, как плохое использование кэша, неудачное размещение данных в памяти и пр. Исправление этих недостатков способно значительно поднять общую производительность вычислителя.

Существуют примеры разработки визуальных сред для представления характеристик работы единичного процессора вместе с представлением традиционных параметров производительности (например, баланса загрузки, эффективности синхронизации). В этих средах также предлагаются интересные метафоры визуализации, порождающие трехмерные виды отображения. Однако эти метафоры не слишком очевидны, что приводит к сложностям интерпретации результатов визуализации.

Проблемы отладки и настройки производительности актуальны не только для параллельных систем, но и для многих современных программных комплексов, функционирующих в распределенных вычислительных средах. Поэтому активно разрабатываются (и используются) соответствующие программные продукты, содержащие необходимые средства визуализации. При этом используются известные подходы - и традиционные виды отображения метрик производительности, и отображения характеристик единичного процессора, и представление некоторых абстракций работы распределенных и параллельных систем, по которым можно оценить «узкие места» производительности, например, представление графа вызовов сложной программы. По необходимости в промышленных средах приходится использовать достаточно апробированные методики визуализации, которые хотя не всегда могут обеспечить требуемую эффективность визуализации и интерпретации, но достаточно надежны и привычны для пользователей.

5. ОБСУЖДЕНИЕ

Регулярный мониторинг интернета с целью обнаружения новых систем визуализации программного обеспечения параллельных вычислений показывает, что с конца XX века произошло достаточно резкое сокращение серьезных работ на эту тему. С трудом нашелся один новый (хотя и весьма интересный) проект визуального языка параллельного программирования. Практически прекратились новые публикации о соответствующих проектах, проводившихся в известных центрах, например, в исследовательских центрах NASA, где в 90-х годах были получены блестящие результаты.

Можно, конечно, считать, что причина заключается в кризисе, возникшем в параллельных вычислениях, и связанном с возможным исчерпанием новых идей в создании параллельных вычислителей с мультикомпьютерной архитектурой. Однако, по нашему мнению основная причина понижения интереса к визуальным параллельным языкам и отладчикам связана прежде всего с проблемой визуального описания абстракций параллельного программирования.

«Мелкозернистое» параллельное программирование, в ходе которого идет кропотливая работа по кодированию и анализу состояния сотен и сотен процессов, плохо отображается в конкретные визуальные объекты. Проблемы возникают даже с размещением на экране сотен образов и навигации в большом количестве графических выводов.

Хуже того. Визуальное представление графа потока управления и/или потока данных не дает нового качества для программиста при описании сущностей параллельного программирования. Важным результатом стала бы разработка таких средств, которые позволили бы описывать динамику параллельных процессов при помощи динамических образов.

Аналог подобных идей можно найти в ранних системах программирования путем демонстраций, где частично определенная программа начинает выполняться, отображая результаты в визуальном виде. У нас есть опыт разработки макета визуального языка параллельного программирования с элементами динамического описания динамики. В этом языке пользователь, используя такие понятия, как процесс, обмен, работа, ожидание, канал и др., и задавая времена работы и ожидания, должен определять основные характеристики процессов в специальных окнах. Затем визуально описываются коммуникационные связи между процессами. Уже на ранних этапах описания процесса начинается динамическое отображение его работы в виде модифицированных диаграмм Гантта. Дополнительные

описания процесса как на внутреннем уровне, так и на уровне взаимодействия изменяют картину, отображающую его деятельность.

Однако замена графового представления параллелизма динамикой диаграмм Гантта не дала серьезного повышения эффективности параллельного программирования.

Другая попытка выйти из «графовых рамок» связана с разработкой средств визуализации для системы параллельного программирования DVM. В случае DVM, где используется параллелизм по данным, распараллеливание программы заключается во внедрении в исходный текст директив, невидимых для обычного компилятора, но разворачиваемых DVM-конвертером в коммуникационные вызовы. Параллелизм по данным означает, что в параллельной программе одни и те же действия осуществляются одновременно над множеством наборов данных. В DVM введена концепция виртуальных процессоров, на массив которых происходит распределение данных (фрагментов многомерных массивов). Для облегчения проектирования программ и отладки правильности были разработаны средства визуализации распределения данных, перед которыми ставились, 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 качестве графического объекта выступала таблица, представляющая вектор или двумерный массив данных. В основе визуального распределителя лежит принцип непосредственного действия, когда манипуляции с графическим объектом приводят к изменению исходного текста программы. Созданные экспериментальные прототипные системы показали возможность решения задачи визуализации распределения для одномерных и двумерных массивов.

С опытом разработки среды визуальной поддержки проектирования и отладки производительности для системы DVM связаны и другие наши наблюдения. Так, при разработке системы видов отображения для отладки эффективности были использованы исключительно двумерные способы представления, такие как иерархический список, радиальная диаграмма, «стена интервалов». Попытки перейти на (казалось бы) более информативные трехмерные образы (например, «войти в стену интервалов», чтобы получить дополнительную информацию о содержимом отдельного интервала) не увенчались успехом - получалось излишнее усложнение как образности, так и навигации в ней. Оказалось, что характер отладочной информации, получаемой в DVM, по своей природе не нуждается в трехмерных отображениях. Однако удачным оказалось применение графики промежуточной (или комплексной) размерности, когда на стенах виртуальной комнаты, соответствующей некоторой программной функции выводятся двумерные пиктограммы, представляющие различные программные объекты. При этом связи между объектами задаются в трехмерном пространстве «комнаты», а связи между функциями и навигация между ними в виртуальном «небоскребе». Причем метафоры комнаты и небоскреба могут использоваться как в целях визуального программирования, так и при отладке производительности в макетной реализации вида отображения графа вызовов. Граф вызовов был представлен как набор связанных между собой помещений. Каждая комната - визуальное представление функции. Кроме того, за функцию «отвечает» и пиктограмма на стене комнаты функции-родителя в графе вызовов. Ребра графа естественным образом переносятся на изображение - они следуют от пиктограммы функции на стенке родителя к комнате-кубу данной функции.

Для представления графа вызовов нами был предложен еще ряд метафор, в том числе метафора молекулы. В этой метафоре структура молекулы отражает структуру исходного графа. Между атомами (отображающими функции - вершины графа) введено два типа взаимодействия: упругое между связанными и электростатическое между всеми вершинами-атомами. Электростатическое взаимодействие отражает временные характеристики вызовов функций, тогда как упругое - количество вызовов. Анимация (например, вращение молекулы) позволяет изучить структуру графа. Анализ возможностей этой метафоры показал, однако, ее большую применимость не для отображения графа вызовов, а для представления структур больших систем. Как и в других случаях при анализе видов отображения, построенных на базе «физической» метафоры, требуется двойная интерпретация, что несколько затрудняет их использование.

Отладка правильности или эффективности программ, да и просто понимание того, как они работают, приводит к необходимости решения очень серьезной задачи. Программа, с одной стороны, представляется статичным набором кода (или неких диаграмм, схем, пиктограмм, глифов и пр. в случае визуального программирования). А с другой - разворачивается во времени в некоторый процесс изменения своего состояния. Причем при каждом новом наборе параметров и входных данных получается разный результат. Перед тем, кто работает с программой, встает задача восприятия хода выполнения разворачивающегося во времени процесса по имеющемуся его статичному или анимированному описанию.

Под описанием процесса подразумевается как текст программы, выполнение которой подлежит изучению, так и данные о состоянии программы на разных этапах ее выполнения.

Можно предложить метафору «траектории», в которой процесс, выполняющийся во времени представляется как траектория в пространстве состояний процесса. Вводится точка выполнения процесса, которая перемещается по траектории, отражая, что делается в каждый момент времени. В случае детерминированного выполнения программы, по исходным данным можно однозначно получить, куда траектория приведет. Траектория позволяет воспринимать не только качественные характеристики (то есть куда пришли), но и количественные - за какое время был совершен переход из одного состояния в другое. Однако пространство состояний получается невероятно сложным, даже для сравнительно счетной небольшой программы. Причина - чрезмерная детализация. Для адекватного восприятия программы (и как исходного текста - закодированного алгоритма, и как хода процесса выполнения) применяется агрегирование, при котором некая последовательность действий представляется на более высоком уровне как одно действие. По аналогии с детализацией и агрегированием фрагментов кода или алгоритма можно агрегировать пространство состояний. Таким образом, данная метафора может использоваться как для описания операций, так и для настройки эффективности в пакетах параллельных вычислений, решающих сложные прикладные задачи. Такие пакеты, простые примеры которых были рассмотрены выше,

должны занять большее место среди сред, обеспечивающих параллельные вычисления.

6. ЗАКЛЮЧЕНИЕ

Подведем итоги обзора и обсуждения состояния дел в области визуализации программного обеспечения параллельных вычислений.

Анализ состояния дел вместе с накопленным собственным опытом экспериментальных разработок позволяет сделать некоторые выводы в связи с визуализацией программного обеспечения параллельных вычислений.

Весьма значительная интеллектуальная сложность параллельного программирования настоятельно требует инструмента представления как абстракций, так и программных объектов на всех этапах разработки параллельных программных комплексов. Поэтому, начиная с 80-х годов прошлого века, средства визуализации активно разрабатываются в различных академических и исследовательских центрах. Однако развитие параллельных вычислений быстро обогнало возможности визуализации, где естественные ограничения по восприятию образов человеком не позволяют выводить данные о требуемом количестве объектов (например, о сотнях и тысячах параллельных процессов). Возможности трехмерной графики и анимации не удалось в полной мере использовать для увеличения выразительности используемой образности. Кроме того, использование графовых нотаций зачастую не дает нового качества при программировании, а среды, построенные на их основе, могут служить лишь вспомогательным средством.

Таким образом, идея о чисто визуальных (или визуальных по преимуществу) средах параллельного программирования видимо не осуществима в полном объеме. Полезными в этой связи могут оказаться хорошо себя зарекомендовавшие визуальные средства поддержки программирования, включая сюда и мастера текстов, использованные нами при генерации директив распараллеливания в системе DVM.

Также вспомогательную роль играет визуализация и в системах отладки правильности и производительности параллельных программ, и это происходит, несмотря на блестящие примеры разработки таких систем в ведущих научно-исследовательских центрах США.

Вероятно, необходимо разделение «мелкозернистого» (традиционного) и «крупнозернистого» параллельного программирования. В первом случае при создании комплексов с использованием MPI, OpenMP или DVM визуализация будет использоваться для поддержки кропотливой работы разработчиков. Во втором случае пользователем параллельных комплексов выступают специалисты, решающие свои большие и сверхсложные задачи с использованием параллельных пакетов с визуальными входными языками.

Здесь необходим выбор эффективных абстракций программирования, поиск метафор визуализации и построения на их основе систем видов отображения.

7. ЛИТЕРАТУРА

[1.] Авербух В.Л. *Средства визуализации параллельного программирования (обзор) // Пользовательский Интерфейс: исследования, проектирование, реализация. 1993, N 4. стр. 32-41.*

[2.] Авербух В.Л. *Визуальная отладка параллельных программ (обзор) // Алгоритмы и программные средства параллельных вычислений. Екатеринбург. ИММ УрО РАН. 1995. Стр. 21-46.*

[3.] Stankovich N., Zhang K. *Visual Programming for Message-Passing Systems // International Journal of Software Engineering and Knowledge Engineering, vol.9, N 4, August 1999, pp.397-423.*

[4.] Zhang K., Hintz T., Ma X. *The role of graphics in parallel program developing // Journal of Visual Languages and Computing. 1999. N 10. pp. 215--243.*

[5.] Авербух В.Л., Байдалин А.Ю. *Разработка средств визуализации программного обеспечения параллельных вычислений. Визуальное программирование и визуальная отладка параллельных программ // Вопросы атомной науки и техники. Сер Математическое моделирование физических процессов. 2003. Вып. 4. С. 68-80.*

[6.] Авербух В.Л., Байдалин А.Ю. *Разработка средств визуализации программного обеспечения параллельных вычислений. Оптимизация программ // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов. 2004. вып. 1. С. 70-80.*

Об авторах

Авербух Владимир Лазаревич, заведующий сектором компьютерной визуализации ИММ УрО РАН, averbukh @ imm.uran.ru

Байдалин Александр Юрьевич, аспирант Уральского Государственного университета, bajur @ imm.uran.ru

Исмагилов Дамир Ришадович, аспирант ИММ УрО РАН

Казанцев Алексей Юрьевич, аспирант ИММ УрО РАН

State of the Art in Software Visualization for Parallel Computing

Abstract

The overview of State of the Art in Software Visualization for Parallel Computing is made.

This discipline includes the visual languages and visual debuggers of correctness and effectivness. Also the discussion and some forecasts of development are made.

Keywords: Software Visualization Parallel Computing.

About the authors

Vladimir L. Averbukh. Ph.D. head of the researcher's section of Institute for Mathematics and Mechanicks Urals Branch of Russian Academy of Science. averbukh @ imm.uran.ru

Alexandr Baydalin, Ph.D. student at Urals State University, bajur @ imm.uran.ru

Damir Ismagilov, Ph.D. student at Institute for Mathematics and Mechanicks Urals Branch of Russian Academy of Science

Alexey Kazantsev, Ph.D. student at Institute for Mathematics and Mechanicks Urals Branch of Russian Academy of Science