УДК 519.6

РАЗРАБОТКА СРЕДСТВ ВИЗУАЛИЗАЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ. ОПТИМИЗАЦИЯ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

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

Работа содержит обзор состояния дел в визуализации программного обеспечения параллельных вычислений и описание разработок авторов в области создания средств визуальной отладки эффективности для отечественной системы параллельного программирования DVM.

Введение

В предыдущей статье [1] рассмотрены проблемы, возникающие при разработке визуальных сред проектирования и отладки правильности параллельных программ.

Сложность параллельного программирования и отладки параллельных программ приводит к тому, что большинство параллельных отладчиков использует визуализацию для представления информации. К настоящему времени в разных странах мира разработаны десятки интересных систем визуализации параллельных вычислений различного назначения, включая разнообразные системы отладки эффективности.

Существует два аспекта разработки систем отладки производительности: в связи с проблемами инженерии программного обеспечения и собственно с визуальными задачами. С первым аспектом связаны вопросы о методах реализации системы сбора данных, расстановки в теле изучаемой программы контрольных точек, очень сложные методики, позволяющие реализовать on-line отладку и т. п. Второй аспект касается того, как, в каком виде представлять данные о производительности. По сути второй аспект (которому уделяется мало внимания по сравнению с первым) оказывается очень глубоким и связывается с вопросом о том, какие сущности параллельного программирования необходимы для отладки производительности. Уже после ответа на этот вопрос можно думать о метафорах визуализации и видах отображения.

Отладка эффективности

Системы отладки эффективности служат для того, чтобы предсказывать, находить и избегать возможной неэффективности исполнения параллельных программ. Несмотря на большое количество инструментальных средств, описанных в литературе, лишь немногие из них реально используются. Для того чтобы разработать хороший инструмент, помогающий программисту отлаживать и повышать эффективность своей программы, необходимо рассмотреть возникающие ошибки и изучить их природу.

Имеет место классификация проблем отладки и настройки эффективности [2]. При этом можно выделить следующие факторы, затрудняющие поиск ошибок:

1) разрыв между причиной и следствием;
2) неточные инструментальные средства;
3) неверное предположение или модель;
4) неструктурированная программа.

Часто проявление ошибки программы далеко от ее причины. Трудно понять, в чем дело, когда ошибку нельзя связать с исходным текстом программы. К этой категории относятся ошибки, которые имеют много "степеней свободы", т. е. являются нечастыми, неустойчивыми, несогласованными. Знание о том, что эффективность достигается оптимальностью, не связано со знанием того, какую часть кода надо оптимизировать. Хотя достаточно просто определить признаки неэффективности, часто очень трудно найти, где локализуется ее причина. Преодоление этого разрыва, как правило, начинается со

сбора данных по трассировке программы, объем которых может быть очень большим (особенно в случае масс-параллельных программ). Сопоставить данные трассировки счета программы на низком уровне трудно. Например, отладчик показывает, что время потрачено, но не ясно, где именно, так как показывается суммарное время для всех процессоров.

Существуют ошибки, которые исчезают (или появляются), когда включены средства отладки — явление, известное как *эффект зонда*. Возможны также *эффекты разрушения* правильного выполнения параллельной программы за счет внедрения отладочного и визуализационного инструментариев.

Модель, объясняющая работу операции программного обеспечения системы или аппаратного компонента, может значительно сократить область поиска причины ошибки, но также в равной степени может все усложнить, если модель неверна.

Иногда отладить чужую программу сложно из-за громоздкости кода, его засоренности.

Большинство систем отладки эффективности 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. Для визуального представления метрик применяются виды отображения, как правило, заимствованные из статистической графики.

Именно на методах статистической графики в основном базируются виды отображения системы ParaGraph [3], ставшей неофициальным стандартом системы отладки эффективности в середине 90-х годов. ParaGraph поддерживает более 25 видов отображения, среди которых — как хорошо известные столбчатые и круговые диаграммы и графики различных типов, так и менее привычные, например диаграммы Гантта или диаграммы Кивиата.

Работа целого ряда систем отладки эффективности, разработанных в первой половине 90-х годов, заключается в сборе данных о производительности параллельной программы для последующей передачи визуализирующей части системы ParaGraph. В других системах отладки эффективности (см. [4], PIE [5], RP3 [6], PEP [7]) содержится примерно тот же набор визуальных средств, который представлен в системе ParaGraph. Из более поздних (середина 90-х годов) разработок, следующих в этом направлении, рассмотрим системы Paradyn, P3T, AIMS и очень интересную систему Avatar, основанную на использовании виртуальной реальности для нужд отладки эффективности, а также системы Kanoko и Chiron.

Paradyn [8] является инструментом измерения производительности очень больших параллельных программ и среди других включает в себя модуль *Консультант эффективности*, который определяет "узкие места" производительности путем автоматического поиска в информационном пространстве, определенном моделью поиска по трем осям: *почему*, *где* и *когда*. (*Почему* данная прикладная программа имеет низкую производительность? *Где* расположено узкое место производительности? *Когда* проявилось воздействие данного узкого места на производительность программы?)

Когда Paradyn подсоединяется к прикладной программе, подсистема управления средствами измерения определяет все потенциальные позиции ввода средств измерения путем сканирования двоичного кода прикладной программы. После этого подсистема управления средствами измерения создает небольшие фрагменты программы (*trampolines*). Эти заготовки вставляются в двоичный текст прикладной программы. Используя средства ОС UNIX, подсистема управления инструментальными средствами обеспечивает функционирование этих заготовок в теле прикладной программы.

Иной подход к отладке эффективности реализован в системе P3T [9], предназначенной для оценки и оптимизации производительности параллельных программ. Если в предыдущих работах рассматривалась отладка эффективности на уровне операций посылки сообщений, то здесь проводится оптимизация самого кода программы при ее распараллеливании. Система P3T вычисляет набор параметров параллельной программы, каждый из которых отражает различные аспекты производительности. С помощью P3T можно ответить на три основных вопроса, касающихся отладки эффективности параллельной программы:

1) какой из аспектов производительности необходимо улучшить;
2) где находятся части программы, требующие увеличения производительности;
3) что должно быть сделано, чтобы увеличить производительность.

Ключевой проблемой при разработке систем оценки производительности является доведение важной информации о поведении выполняемой программы как до программиста, так и до компилятора. Это позволяет организовать настройку эффективности программ. В системе реализован набор интерактивных инструментов, служащих для настройки эффективности и визуализации данных о производительности прикладной программы. Отображаются параметры параллельной программы, а их изменение пользователем обеспечивает выбор оптимальных стратегий программирования, связанных с распараллеливанием. Визуализация производительности осуществляется для единичного выражения. В окне визуализации в виде цветных полос выдаются параметры параллельной программы. Их значения сравниваются со значениями параметров выполнения, показавшего наихудшую производительность, выведенными в нижней части окна.

Пользователь имеет возможность выбрать стратегию распределения по процессорам структур данных (например массивов) и распараллеливаемых программных конструкций, таких как циклы.

Система AIMS [10] является инструментарием для настройки и предсказания производительности программ, основанных на посылке сообщений (работающих как в сети, так и в рамках параллельного вычислителя). AIMS содержит набор подсистем, служащих для измерения и анализа производительности.

Измерения, проведенные в ходе выполнения программы для создания файла трассировки, нарушают и коммуникационные, и временные характеристики событий, записанных в файле. Для сглаживания последствий этих нежелательных эффектов служит специальный модуль компенсации, использующий соответствующие модели оценки взаимодействий для данной вычислительной архитектуры. Компенсатор внедрения исправляет файл трассировки, который может затем служить входным файлом для разнообразных средств оценки производительности, определяющих характеристики производительности на основании *посмертных* данных.

Визуализация производится с использованием анимационных видов отображения. Эти виды отображения представляют информацию, указывая, когда выполняются отдельные программные конструкции, когда посылаются сообщения, как долго сообщения стоят в очереди перед обработкой, когда простаивают отдельные процессоры. Некоторые виды отображения показывают фрагменты истории программы, непрерывно прокручивая изображение, другие, обновляя предыдущее состояние, создают анимационную последовательность образов.

Визуальная среда Avatar [11], созданная в рамках проекта Pablo, базируется на использовании средств виртуальной реальности и предназначена для представления очень больших объемов данных о производительности параллельных систем, которые она получает непосредственно в ходе их работы. Avatar также обеспечивает адаптивное управление прикладными программами в реальном времени.

Основной визуальной метафорой системы Avatar является *Scattercube-матрица*. Scattercube-матрица представляет собой трехмерное обобщение используемых в статистической графике отображения *scatterplot*, *метода множественного вывода данных* и *scatterplot-матрицы*. При работе с системой пользователь, применяющий средства виртуальной реальности, как бы оказывается внутри трехмерного помещения, в котором на внутренних гранях показаны оси, а на гранях куба — полу и стенах помещения — выводятся кривые, описывающие поведение параллельной программы. Набор единичных scattercube'ов напоминает стеклянный небоскреб, каждая из комнат которого содержит трехмерный вывод, описывающий различные аспекты поведения параллельной программы. "Путешествие по небоскребу" дает возможность полного исследования данных о производительности прикладной программы.

Пользователь "погружен" в виртуальную реальность, описывающую поведение параллельной программы. Он осуществляет управление выводом информации, а также адаптивный контроль работы самой прикладной программы. Существует несколько конфигураций системы Avatar на базе рабочей станции, снабженной монитором, ручного манипулятора с шестью степенями свободы, шлема с встроенным экраном и *театра виртуальной реальности* CAVE.

Сцена театра CAVE первоначально представляет собой куб размером с комнату, где на грани (стены) проецируется видеоизображение с высокой разрешающей способностью. Avatar использует звук для подкрепления эффективности вывода информации и увеличения числа размерностей выводимых данных за счет невизуального представления значений полученных оценок.

Кроме того, звук используется в целях перемещения, ориентировки и взаимодействия в виртуальном пространстве. При помощи характеристик звучания (например высоты звука) описываются статистические характеристики полученных оценок. По мнению разработчиков, система обеспечивает четкое визуальное различие между эффективной и неэффективной конфигурациями прикладной программы при примененном виде отображения. Накоплен большой опыт в области интерактивной интерпретации данных, ориентации и интерактивного управления перемещением в виртуальном пространстве.

В анимационной системе Kanoko [12] предлагается метод анимации, отображающий значения состояния программы, взятые из ее трассы, в набор состояний динамической модели системы. Затем воспроизводится работа параллельной программы (моделируется динамическая система), а результаты моделирования визуализируются (а также сонифицируются). Анимация представляет изменения в балансе между вычислениями и обменами как обычные движения некоторой динамической системы. (Таким образом, имеет место понятная для пользователя метафора.) В частности, определяются используемая динамическая система и отображение элементов и состояний параллельной ЭВМ на тела и силы динамической системы. Моделирование параллельной ЭВМ ведется на основе топологии ее (физической) сети. Так, для параллельного компьютера вычислительные модули и производимые на них вычисления, коммуникационная сеть и количество обменов могут отображаться соответственно на тела и их массу, пружины, связывающие тела, и силы притяжения между телами.

Визуальная система отладки эффективности параллельных программ Chiron [13] отображает эффективность работы программ многопроцессорных машин с общей памятью. Здесь используется трехмерная графика, что позволяет показывать большое количество информации. Chiron разработан для решения таких проблем, как плохое использование кэша, неудачное размещение данных в памяти, разделение работы между процессорами, неэффективность синхронизации.

Проект Chiron поддерживает три вида отображения данных, каждый из которых представляется отдельным объектом в трехмерном пространстве и может управляться в интерактивном режиме (вращением, переключением уровня детализации, изменением масштаба):

1. *Глобальный*, представляющий краткий обзор интересующего пользователя показателя для всех объектов, классов или выражений программы. Система поддерживает два показателя: стоимость обращения к памяти и стоимость синхронизации (включая время ожидания). Глобальный вид отображения представляет собой трехмерный свернутый граф. Объекты сортируются согласно некоторому критерию (показателю), затем они помещаются последовательно вдоль диагоналей квадратной сетки, в результате чего получается поверхность.
2. *Отображение корреляций*, полезное для понимания того, как некоторые свойства всего набора объектов соотносятся со свойствами определенного объекта, или подмножества объектов, или участка программы. Отображение корреляции включает проецирование дополнительного представляющего интерес показателя на поверхность свернутого графа (т. е. поверх глобального вида) либо с помощью расцветки текстуры на поверхности, либо добавлением дополнительного показателя как малого возмущения основной поверхности.
3. *Временное отображение.* Обеспечивается несколько типов временных отображений:
   - размещения кэша — показывает, как объекты и блоки кэша пространства виртуальной адресации перемещаются из различных кэшей;
   - активности процессора — показывает, с каким объектом какие процессоры обращаются в данное время;
   - температуры объекта — приблизительно соответствует производной отображения кэша.

Система Chiron в основном позволяет исследовать влияние работы одного из процессоров на общую производительность параллельного вычислителя. Отметим также, что, как и в системе Kanoko, в ней используются новые интересные метафоры визуализации (*температура объекта*, *динамическая система*) для представления и анализа данных о производительности процессоров.

Разработки немецкой фирмы Pallas Vampir Analyzer и GuideView [14] предназначены для анализа производительности параллельных программ.

Изначально Vampir Analyzer применялся для анализа программ, написанных с использованием MPI. Сбор временных характеристик осуществляется как путем инструментирования изучаемой программы за счет внедрения дополнительного слоя между программой и коммуникационной библиотекой, так и с использованием комплекса Hardware Perfomance Analysis. Накопленная статистика представляется как в текстовом, так и в графическом виде: круговые диаграммы, диаграммы Гантта, карты обменов и др.

GuideView — программное средство, изначально развивавшееся само по себе, но с некоторого момента объединенное с Vampir Analyzer. GuideView предназначено для сбора и анализа параллельной трассы. Сбор данных производится за счет инструментирования программы и обращения (во время выполнения) к специализированной библиотеке поддержки. Отличительной особенностью системы VGV (Vampir/GuideView) является перспектива промышленного использования.

В настоящее время комплекс Vampir Analyzer/GuideView позволяет анализировать как программы, основанные на парадигме передачи сообщений, так и программы на основе общей памяти (OpenMP).

В 80-е годы системы визуализации параллельных вычислений базировались на посмертной визуализации собранных данных о работе параллельной программы. В 90-е годы реализуются интерактивные визуальные отладчики параллельных программ для отладки как их правильности, так и эффективности. Примерами последних могут служить специальный модуль компенсации, реализованный в системе AIMS для создания эффекта интерактивной отладки, или trampolines системы Paradyn. Однако следует отметить некоторую искусственность и несомненную трудоемкость и громоздкость способов создания эффекта *живой* визуализации.

Проблема разработки видов отображения связана с другой нерешенной проблемой — эффективной пользовательской интерпретацией отладочных данных. При этой интерпретации следует опираться либо на привычные и апробированные формы представления данных, либо на метафоры, созданные на основе опыта пользователей и учитывающие особенности изучаемых программных объектов.

Заимствование из статистической графики видов отображения на базе диаграмм Гантта, Кивиата, столбчатых и круговых диаграмм, графиков, гистограмм и их трехмерных модификаций позволяет использовать накопленный опыт представления данных. Однако, используя эти методики, сложно получить интегральное представление о производительности масспараллельной системы.

Несмотря на применение в системе Avatar достаточно экзотических методик, основанных на виртуальной реальности, сам репертуар видов отображения, используемых при визуализации, ограничен теми же графиками, представляющими метрики производительности. Более того, по мнению авторов, пользователь может испытывать затруднения как при интерпретации десятков и сотен графиков, так и при навигации в рамках виртуальной реальности с использованием несовершенных методов взаимодействия.

Неоднозначно могут оцениваться и попытки разработки видов отображения на базе новых метафор. Искусственные и/или случайные сближения могут дать интересные, но не слишком легко и однозначно интерпретируемые картинки. (О новых метафорах в отладке производительности см. ниже.)

В этой связи может помочь комплексный подход к разработке систем параллельного программирования, когда полноценно и в едином ключе реализованы средства описания параллельных программ и их отладки правильности и производительности. Тогда, опираясь на анализ основных программных сущностей системы, можно спроектировать специализированные виды отображения как для спецификации программ, так и для представления отладочных данных. Именно такой подход реализован при проектировании системы DVM.

Визуальные средства анализа эффективности в системе VProjDVM

Система DVM [15] снабжена развитыми средствами сбора отладочной информации, включая анализатор эффективности и предсказатель производительности.

Единицей отладочной информации для системы DVM является интервал — фрагмент кода, выделенный системой или заданный пользователем, выполняемый на конкретном процессоре. Следует отметить, что интервал — сущность динамическая. К примеру, система позволяет рассматривать как интервал отдельную итерацию цикла, вызов функции или любой другой блок операторов. Временные характеристики интер-

вала включают в себя не только общее время выполнения, но и такие важные его составляющие, как время вычислений, время, потраченное на операции обмена данными, время простоя процессора. Сбор отладочной информации может осуществляться для каждого из процессоров и производится за счет внедрения средств сбора информации (программных закладок) в используемые программой библиотеки поддержки времени выполнения.

Большинство видов отображений используется для представления простых по своей природе данных, например набора скалярных величин. Основная сложность их применения для визуальной отладки DVM заключается в том, что характеристики, собираемые подсистемой и используемые для отладки эффективности, являются комплексными, состоящими, возможно, также из комплексных компонентов. Отображение каждого из компонентов в отдельности либо потребует от пользователя весьма напряженной работы мысли по созданию и удержанию в голове связей между всеми этими величинами, либо приведет к неполной и/или искаженной интерпретации. При этом нужно помнить еще и то, что DVM является системой параллельного программирования и отладочная информация собирается для нескольких (от единиц до сотен) процессоров. Традиционно в визуальных отладчиках либо используются общеизвестные простые виды отображений, присущие стандартному текстовому отладчику, либо разрабатываются свои собственные.

В последнем случае основную сложность может представлять само взаимодействие пользователя со средствами отладки, например по той причине, что у него могут быть внутренние представления программных объектов, отличающиеся от предлагаемых. Для создания эффективного и удобного в работе визуального отладчика требуется не просто создавать виды отображений, необходимо изучить структуру визуализируемой информации и при разработке новых специальных видов отображений отталкиваться от внутреннего устройства, взаимодействия и функционирования визуализируемых сущностей. При этом крайне желательно использовать не только опыт, накопленный при разработке визуальных отладчиков, но и пользоваться средствами информационной визуализации, которые позволяют представлять в воспринимаемом для человека виде весьма сложные информационные структуры.

В случае DVM структура визуализируемых данных и методика работы с ними у пользователя уже сформированы. Поэтому для обеспечения удобства работы было принято решение разрабатывать виды отображения, исходя из сложившейся методики: анализа численных характеристик (с одной или нескольких сторон) и последующего перехода к породившему их (охарактеризованному ими) фрагменту текста.

Чтобы анализировать численные данные, был создан ряд видов отображения для представления характеристик DVM-программ. На базе диаграммы интервала, представляющей единицу статистики, строятся более сложные виды отображения:

- *стена процессоров* — для представления характеристик этого интервала на множестве процессоров;
- *радиальная диаграмма* — для анализа сбалансированности загрузки процессоров при выполнении интервала.

Как уже отмечалось, интервалы могут быть (и являются) вложенными. Для отображения структуры вложенности интервалов и обеспечения навигации по исходному тексту были созданы *список* [*диаграмм*] *интервалов* и *дерево* [*диаграмм*] *интервалов*. Эти виды отображения представляют схему вложенности интервалов соответственно в виде дерева и иерархического списка, в котором подчиненные (вложенные) элементы изображаются с отступом относительно главенствующего. Дерево и список обладают интерактивностью и могут (по воздействию пользователя) разворачивать/сворачивать список своих подынтервалов.

Кроме характеристик выполнения интервала по процессорам, определяются характеристики интервала в целом — общие (средние) и сравнительные (наибольший, наименьший). Эти интервальные характеристики представляются на диаграммах интервалов, размещаемых в узлах дерева или списка.

Для навигации в исходном тексте и структуре интервалов были реализованы следующие решения по взаимодействию пользователя с общей диаграммой интервала (из списка или дерева), обеспечивающие:

- переход к исходному тексту путем подсвечивания первой строки интервала в окне редактирования исходного текста;
- вызов (показ) окна со стеной процессоров или радиальной диаграммой;

– вызов (показ) окна с таблицей исходных численных характеристик.

Вызов окна представления характеристик по процессорам или окна таблицы может быть осуществлен для каждого интервала программы.

В результате взаимодействия с пользователями ИММ УрО РАН были созданы дополнительные виды представления, не связанные (не обусловленные), на первый взгляд, с особенностями DVM-статистики.

Для анализа особенностей работы программы на различных процессорных конфигурациях (DVM-топологиях) был разработан специализированный вид представления — *пучок деревьев*, показывающий в одном масштабе или в пропорциональном виде набор деревьев интервалов.

В дополнение к пучку деревьев для изучения зависимости эффективности также созданы средства построения графиков характеристик пусков в целом одной программы и ее отдельных интервалов на различном числе процессоров. Ведутся работы по изучению возможности автоматизации поиска наилучшей конфигурации.

В результате разработки проекта VProjDVM была получена система видов отображения данных о производительности. В рамках этой системы обеспечено взаимодействие между видами отображения, навигация по структуре данных и просмотр первичных числовых значений. Связь видов отображения с исходным текстом позволяет пользователю быстро выявлять узкие места в параллельной программе (рис. 1–7).

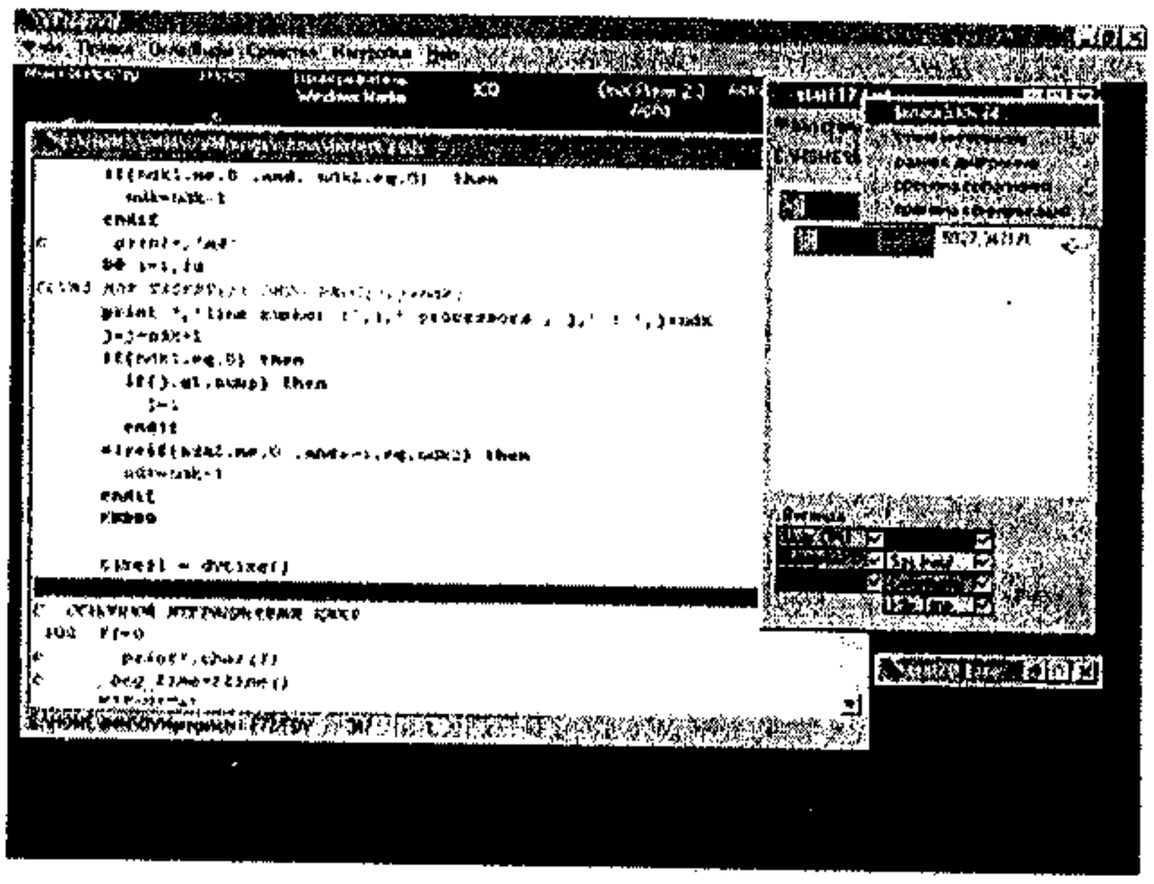

Рис. 2. Подсветка строки интервала, выбранного в иерархическом списке

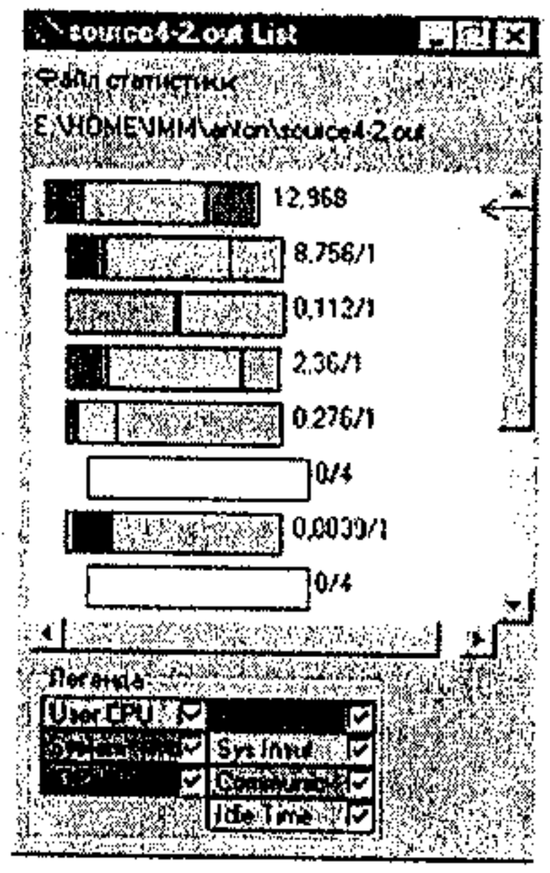

Рис. 3. Список интервалов

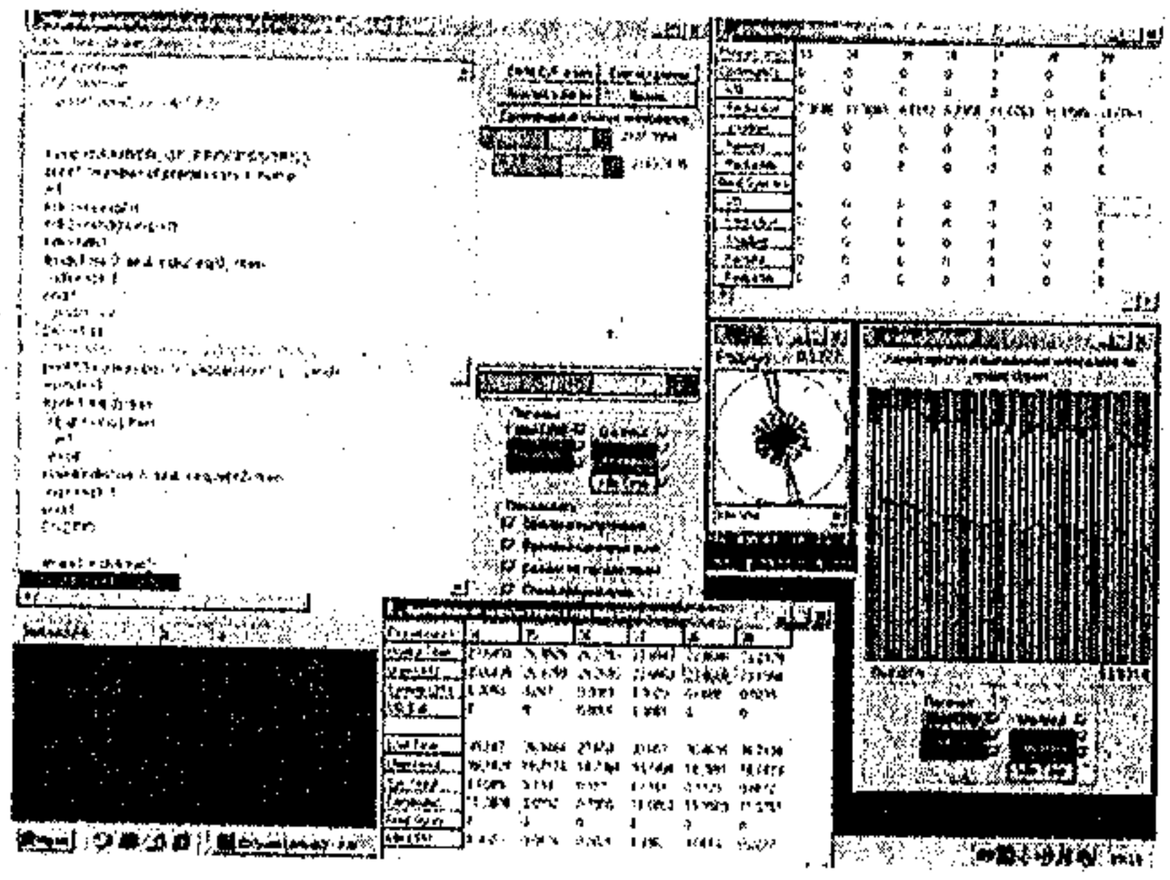

Рис. 1. Общий вид визуализатора при развернутых окнах всех основных средств представления

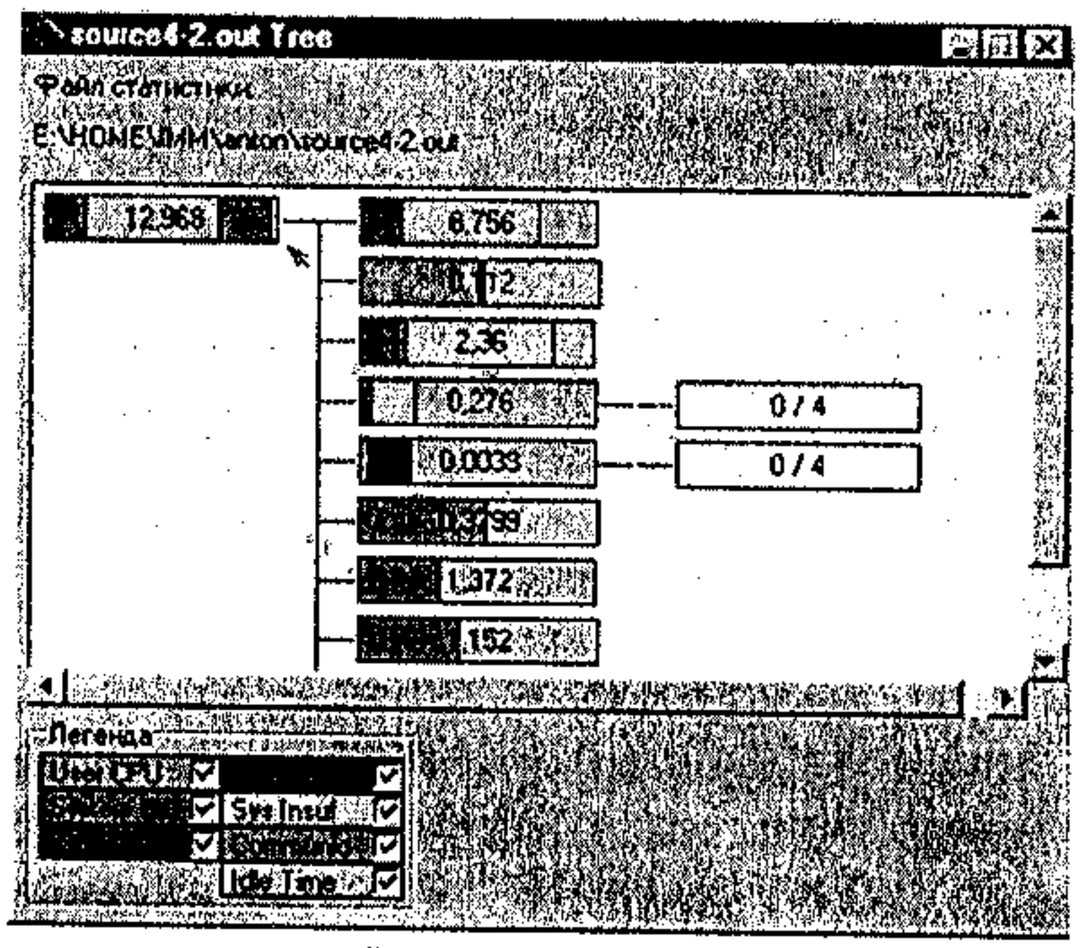

Рис. 4. Дерево интервалов

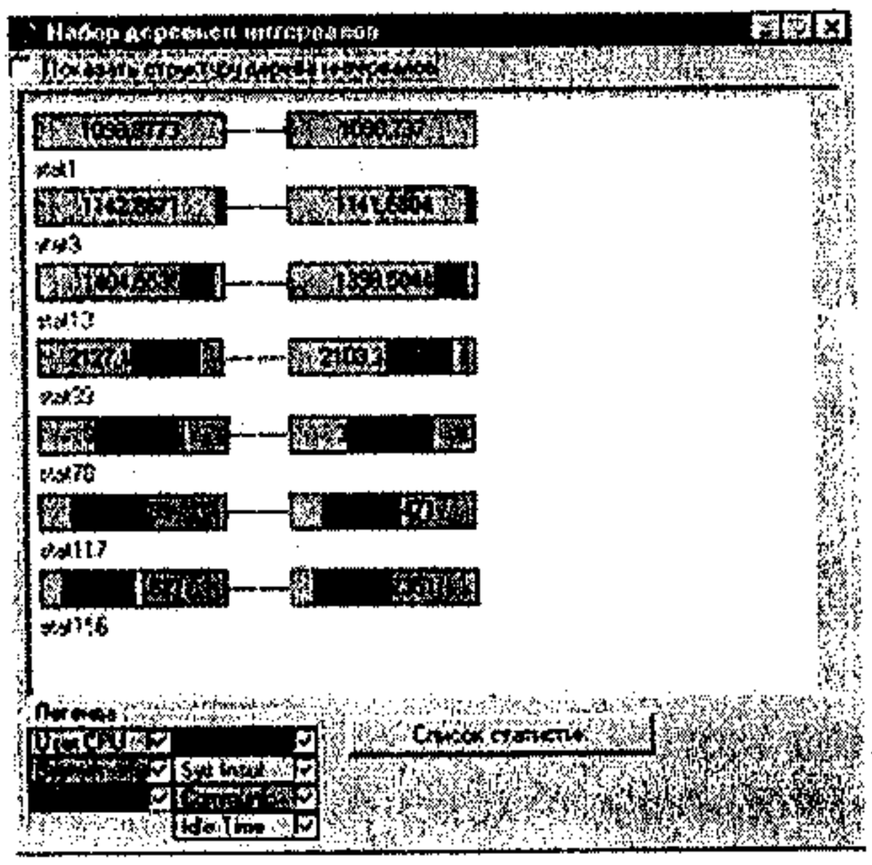

Рис. 5. Набор деревьев интервалов

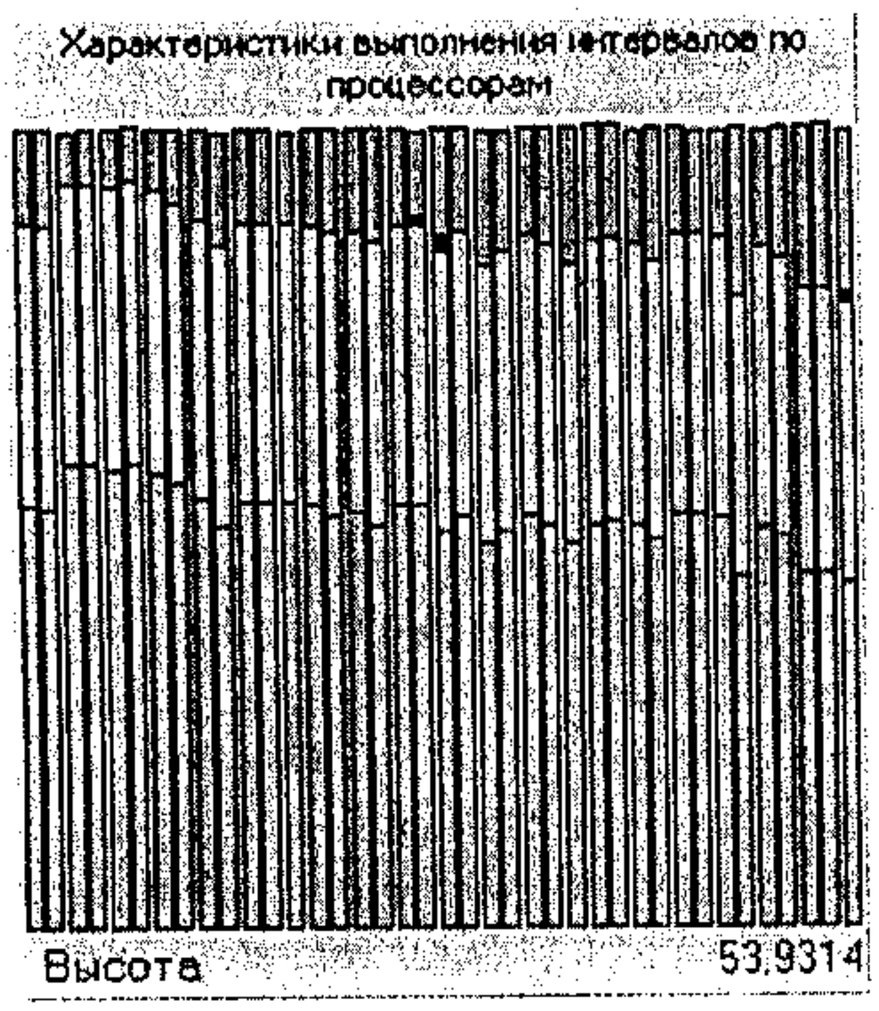

Рис. 6. Стена интервалов

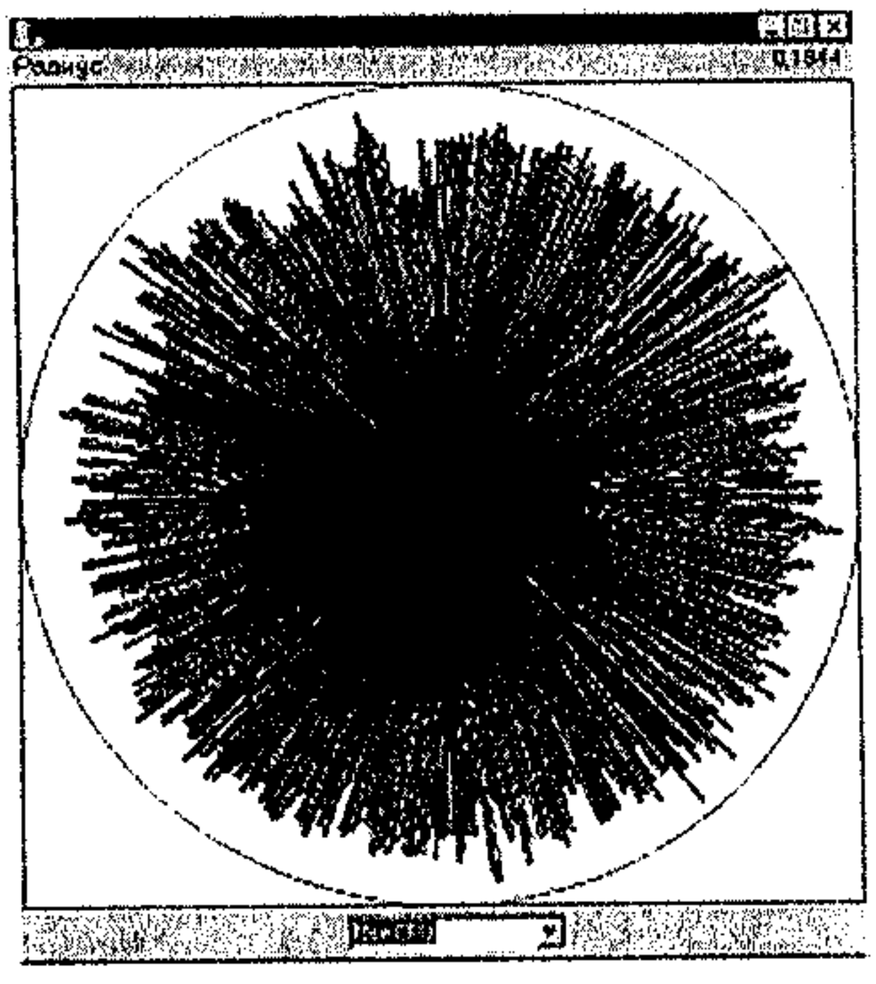

Рис. 7. Радиальная диаграмма

Разработка новых метафор визуализации

Предложенные в предыдущей работе [16] авторские подходы к определению понятия метафоры визуализации позволяют избавиться от необходимости скрупулезного следования зачастую случайным сближениям сущностей, что происходит при традиционном понимании метафоры. В то же время подход, предусматривающий выбор метафор и языков, адекватных в визуализации, объединяет традиционное понимание метафоры человеко-машинного интерфейса и другие случаи использования аналогий и сближений между сущностями при визуализации. Укажем на перенос метафор и видов отображения из информационной визуализации в системы визуализации программного обеспечения. Обратим внимание, что, в свою очередь, системы информационной визуализации заимствовали эти виды из статистической графики.

Метафора порождает некоторое метафорическое пространство за счет того, что объектам целевой области ставятся в соответствие объекты из области источника. А точнее, структурам и/или свойствам объектов из целевой области ставятся в соответствие структуры и характеристики объектов из области источника. Метафора визуализации является отображением (точнее оператором) на некоторый мир визуализации, где безóбразные объекты получают свое визуальное представление. Цель метафоризации состоит в увеличении выразительности изучаемых объектов. При метафоризации выбираются объекты целевого множества с набором структур, свойств и пр., которые желательно рассмотреть с повышенной выразительностью. Причем выбираются не все объекты (и даже не все их свойства или элементы структуры), а лишь те, которые интересуют больше всего. Этим объектам ищутся аналоги (в плане структур, качественных свойств и пр.) в исходном множестве. Далее имеет место следующая операция. Объект целевого множества вместе с объектом из исходного множества помещается в метафорическое пространство, точнее, тем самым порождается метафорическое пространство. В этом пространстве теперь начинает функционировать изучаемый объект. (Можно считать, что это уже новый объект нового пространства.) Метафорическое пространство приобретает автономность от породивших его областей. Многие свойства его объектов лишь опосредованно связаны (если во-

обще связаны) со свойствами объектов исходного множества. Появляется своя логика развития метафорического пространства. В метафоре визуализации метафорическое пространство имеет визуальную природу. Однако при изучении визуализации зачастую приходится сталкиваться с очень сложными и абстрактными метафорами, в которых, на первый взгляд, очень трудно выявить метафорическую сущность.

При проектировании уже упоминавшихся средств визуализации данных о производительности системы DVM были адаптированы несколько видов отображения (и соответственно метафор) информационной визуализации, в том числе информационная стена [17]. В результате получен один из видов отображения системы — *стена интервалов* (правильнее *забор интервалов*). Имела место порожденная реальностью исходного множества идея проникновения в интервал, чтобы получить доступ к его "внутренности" (коду), но эта идея оказалась ненужной, так как при указании на интервал текст кода выводится в специальном окне.

Выше уже упоминалась метафора *комнаты*, которую неоднократно пытались использовать как метафору человеко-компьютерного взаимодействия. С большим успехом эта метафора была неявно применена в системе Avatar для представления данных о производительности большой параллельной программы. (В системе Avatar применяется метафора scattercube, но каждый куб по сути является комнатой, на стенах которой выводятся метрики производительности.) В работах [18—20] описаны прототипные системы и примеры визуализации, в которых метафора комнаты использовалась для создания трехмерных средств визуального программирования, когда на стенах комнаты размещаются иконы (пиктограммы). Эти иконы представляют управляющие конструкции и данные программной функции. Связи между конструкциями показываются в пространстве, а не на плоскости, как в традиционных двумерных диаграмматических и иконических системах визуального программирования.

В рамках системы анализа производительности больших программ VTune, разработанной в фирме Intel, большое внимание уделяется визуализации графа вызовов функций (call graph), используемого для настройки производительности сложных программных комплексов, в том числе распределенных и параллельных программ. В системе VTune используется двумерное представление графа вызовов. При анализе большой по объему и сложной по структуре программы с большой глубиной вложенности вызовов функций и большим количеством пользовательских функций возникают сложности в двумерном отображении протяженной структуры на экране монитора и в анализе пользователем конечного изображения.

Для трехмерного представления этой сущности сделана попытка использования нескольких метафор.

Прежде всего применена метафора комнаты. Граф вызовов 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 стене родителя к комнате-кубу данной функции. Наибольшую реалистичность изображения дает вид отображения "изнутри" комнаты в сочетании с возможностью *путешествия* внутри "здания" между комнатами. Вместе с тем метафора комнаты не является универсальной при визуализации. Кроме того, не решена проблема размещения на экране большого количества комнат, соответствующих серьезной программе.

Следующий вариант представления графа вызовов функции основывается на метафоре *вложенных шаров*, каждый из которых представляет функцию, используемую в программе, как системную, так и пользовательскую. На экране постоянно отображается лишь один уровень вложенности — те процедуры и функции, которые вызываются из текущей. Таким образом удается избежать проблемы самопересечений и нехватки экранного пространства. Переход от функции к функции осуществляется простым нажатием клавиши мыши на том шарике, который представляет необходимую пользователю функцию. Картинка анимирована и интерактивна, что позволяет пользователю лучше понять и проанализировать полученное графическое отображение.

Кроме того, для представления графа вызовов была предложена 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. (Такой подход несколько напоминает идеи, изложенные в [12].) Созданы методы отображения рекурсивного вызова функции. В программной реализации метафоры удалось добиться размещения на экране сотен объектов, "атомов молекулы". Получены интересные примеры отображения графов вызовов, включая представление самой программы визуализации. При этом нарушения оптимальной производительности могут отражаться в виде нарушения четкости структуры молекулы.

Анализ возможностей этой метафоры показал ее применимость не только для отображения графа вызовов, но и для представления структур больших систем. Примечательно, что при изучении структуры данных пользователю, в общем, не важно, какую из органических молекул напоминает сложный трехмерный граф.

Отметим, что всякий раз метафоры визуализации появляются в результате длительного поиска или "озарения" (insight), но не в результате формального порождения. При проектировании метафор вовсе не обязательно заимствовать идеи сближения только лишь из повседневного опыта проектировщика. Опыт реализации средств визуализации для системы параллельного программирования DVM показал, что переработка методов информационной визуализации дает хороший эффект в системах визуализации данных о производительности параллельных систем. Поиск подходящей для пользователя метафоры может потребовать значительных усилий от всех сторон, участвующих в разработке. Особенно сложно найти метафору визуализации и соответствующую систему видов отображения в тех случаях, когда для данной прикладной области не удается подобрать естественную или привычную образность, тем более, когда она вообще отсутствует.

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

Список литературы

1. *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.
2. *Hondroudakis A., Procter R.* An empirically derived framework for classifying parallel program performance tuning problems // Proc. of the SIGMETRICS Symposium on Parallel and Distributed Tools. ACM Press, August 1998. P. 112—121.
3. *Heath M. T.* Performance visualization with ParaGraph // Proc. Second Workshop on Environments and Tools for Parallel Sci. Comput. SIAM, Philadelphia, 1994. P. 221—230.
4. *Авербух В. Л.* Визуальная отладка параллельных программ (обзор) // Алгоритмы и программные средства параллельных вычислений. 1995. С. 21—46.
5. *Lehr T., Seggal Z., Vrsalovich D. F., Caplan E., Chung A. L., Fineman Ch. E.* Visualizing Performance Debugging // IEEE Computer. 1989. Vol. 22, No 10. P. 38—51.
6. *Kimelman D. N., Ngo T. A.* The RP3 program visualization environment // IBM J. Res. Develop. 1991. Vol. 35, No 5/6. P. 635—651.
7. *Szelenyi F., Zecca V.* Visualizing parallel execution of FORTRAN programs // Ibid. Vol. 35, No 1/2. P. 270—282.
8. *Miller B., Calaghan M., Cargille J., Hollingsworth J., Irvin R. B., Karavanic K., Kunchithapadam K., Newhall T.* The Paradyn parallel performance measurement tool // IEEE Computer. 1995. Vol. 28, No 11. P. 37—46.
9. *Fahringer Th.* Estimating and optimizing performance for parallel programs // Ibid. P. 46—56.
10. *Yan J., Sarukkai S., Mehra P.* Performance measurement, visualization and modeling of parallel and distributed programs using AIMS toolkit // Software — Practice and Experience. 1995. Vol. 25, No 4. P. 429—461.
11. *Reed D., Scullin W., Tavera L., Shields K., Elford Ch.* Virtual reality and parallel systems performance analysis // IEEE Computer. 1995. Vol. 28, No 11. P. 57—67.
12. *Osawa N.* An enhanced 3-D animation tool for performance tuning of parallel programs based on dynamic models // Proc. of the

SIGMETRICS symposium on parallel and distributed tools. 1998. Welches, Oregon, US. August 3—4, 1998. P. 72—80.

13. *Goosen H. A., Hinz P., Polzin D. W.* Experience using the Chiron parallel program performance visualization system. 1997. http://www.nhse.org/rib/repositories/ptlib/objects/Asset/chiron.html
14. *Krotz-Vogel W.* Pallas tools for parallel program development // Pallas GmbH MEW'01, Daresbury, 02001-11-28. esc.dl.ac.uk/Cdrom-s/Summer_ School_ 2000/Pallas_ linux.pdf
15. *Коновалов Н. А., Крюков В. А.* Параллельные программы для вычислительных кластеров и сетей // Открытые системы. 2002. № 3. С. 12—18.
16. *Авербух В. Л.* Метафоры визуализации // Программирование. 2001. № 5. С. 3—17.
17. *Jerding, D. F., Stasko J. T.* The information mural: a technique for displaying and navigating large information spaces // IEEE Trans. Visualization and Computer Graphics. Vol. 4, No. 3. P. 257—271.
18. *Исмагилов Д. Р., Шарпан С. В.* Метафора комнаты. Примеры разработки систем компьютерной визуализации // Проблемы теоретической и прикладной математики. Тр. 34 Региональной молодежной конференции. Екатеринбург: ИММ УрО РАН, 2003. С. 264—268.
19. *Байдалин А. Ю., Гусева Е. М., Даеничева А. А., Исмагилов Д. Р., Казанцев А. Ю., Манаков Д. В.* Разработка средств визуализации для системы параллельного программирования DVM // Там же. С. 236—240.
20. *Авербух В. Л., Байдалин А. Ю., Исмагилов Д. Р., Казанцев А. Ю., Поддубная С. В.* Трехмерное представление графа вызовов функций // Тез. Межд. семинара "Супервычисления и математическое моделирование". Саров, 6—13 октября 2003 г. РФЯЦ—ВНИИЭФ: Саров, 2003. С. 12—13.