Задачи визуализации программного обеспечения параллельных и распределенных вычислений

В.Л. Авербух, О.Г. Анненкова, М.О. Бахтерев, Д.В. Манаков
ИММ УрО РАН, УрФУ

В данной работе обсуждаются теоретические и практические результаты развития этой дисциплины за последнее десятилетие. Также рассматриваются проблема формализации и/или верификации визуализации, в том числе в рамках теории принятия решений. Авторами рассматривается задача проектирования средств визуального сопровождения процессов разработки, анализа и отладки программного обеспечения. Приводятся примеры визуализации, используемой в процессе разработки системного программного обеспечения нижнего уровня для современных процессоров с параллельной архитектурой.

1. Введение

Под визуализацией программного обеспечения понимается совокупность методик использования компьютерной графики и средств человеко-машинного взаимодействия, применяемых для спецификации и представления программных объектов и сущностей в процессе создания, отладки и анализа программ, а также для эффективной эксплуатации программного обеспечения. Это направление оформилось как самостоятельное в конце 80-ых годов XX века. Использование визуализации в области параллельных вычислений началось примерно в это же время.

В целом ряде обзоров (см., например, [1-4]) были рассмотрены результаты развития визуализации программного обеспечения параллельных и распределенных вычислений на начало XXI века. В данной работе обсуждаются теоретические и практические результаты развития этой дисциплины за последнее десятилетие. В течение этого времени мы проводим регулярный мониторинг публикаций по данной тематике и собрали значительный корпус публикаций. Исходя из их анализа, а также опираясь на наш опыт исследований и опытных разработок, мы будем строить обзор и попытаемся определить задачи на ближайшее время. Наши подходы к проектированию средств визуализации программного обеспечения связаны с актуальными на данный момент задачами разработки системного программного обеспечения нижнего уровня для процессоров, в которых параллелизм реализован на уровне архитектуры. Отсюда вытекает необходимость разработки средств визуального сопровождения процессов разработки, анализа и отладки программного обеспечения. Мы находимся на первом этапе решения поставленных задач, но уже есть примеры визуализации объектов, возникающих в процессе компиляции программ с языка низкого уровня. Начато также проектирование визуализации для отладчика, разрабатываемого для данного процессора. Визуализация программного обеспечения рассматривается нами как подобласть общей дисциплины – компьютерной визуализации. В связи с развитием компьютерной визуализации как самостоятельной дисциплины ставится вопрос о ее теоретической базе. Важной задачей теории визуализации является создание научных оснований для качественного и надежного проектирования, разработки и оценки визуальных интерактивных систем. Существует несколько подходов к формированию теории визуализации. В данной работе рассматриваются подходы, основанные на формализации и/или верификации визуализации, в том числе в рамках теории принятия решений.

2. Характеристика визуализации программного обеспечения параллельных вычислений

Визуализация программного обеспечения параллельных вычислений подразделяется на три направления – визуальные языки параллельного программирования, визуальные отладчики

правильности параллельных программ, средства отладки (настройки) производительности (эффективности) параллельных программ.

При создании систем визуализации выделяются такие составляющие, как инженерия программного обеспечения; компьютерная графика и/или средства организации человеко-компьютерного интерфейса; аспекты, связанные с “человеческим фактором”.

В нашем случае, “человеческий фактор” касается в основном задач проектирования адекватных видов отображения, использованных в средах визуализации, а также проблем восприятия больших объемов сложно структурированных визуальных данных.

Как мы уже указывали, достаточно подробные обзоры состояния дел в визуализации программного обеспечения параллельных вычислений были сделаны в работах [1-4]. В нашей работе [5] приводится анализ развития визуальных языков параллельного программирования, используемых при разработке программного обеспечения. В данной работе мы отмечаем некоторые характеристики этапов развития сред визуализации программного обеспечения параллельных вычислений, отмечая важные на наш взгляд результаты.

Разработки в области визуализации программного обеспечения параллельных вычислений начались в 80-ые годы. С самого начала разрабатывались языки визуального программирования, первые визуальные отладчики правильности для параллельных вычислений. Мы в своих обзорах ранних разработок основное внимание обращали на предлагаемые виды отображения, так как системная и графическая составляющие этих систем устарели.

Среди особенностей видов отображения этого периода следует отметить использование анимации при отладке правильности. Причем используется как абстрактная образность, так и естественная, отражающая объекты моделей реального мира (как, например, в [6]).

Уже опыт использования систем 80-ых годов выявил проблемы, возникающие при визуализации сравнительно больших объемов информации. Тогдашние системы хорошо показывали работу двух-трех десятков процессов, но отображение работы порядка ста процессов давало весьма запутанные и сложно интерпретируемые картинки.

В 90-ые годы для представления данных об отлаживаемой программе также разрабатывались новые виды отображения. Хотя в большей мере использовались временные диаграммы различных типов. Это связано с важной ролью временных характеристик при организации параллелизма. В системах отладки производительности в основном используются виды графики и диаграммы, заимствованные из статистической графики. Продолжалась разработка новых визуальных языков параллельного программирования, которые строились на основе различных диаграмматических методик.

В качестве важных результатов следует отметить решение проблемы детерминированности в параллельном программировании в рамках визуального языка Phred [7], а также создание системы сравнительной отладки Guard [8]. В этой системе виды отображения строятся за счет сравнения результатов, о которых заведомо известно, что они функционируют правильно (как правило, полученных на базе последовательного варианта программы) и данных отлаживаемых программ. Виды отображения строятся путем наложения эталонных изображений на отображения, полученные на базе нового счета. Такие методики визуализации позволяют пользователю изучать различия в структурах данных.

Однако основное внимание в этот период при проектировании систем визуализации уделялось проблемам инженерии программного обеспечения. Это связано с использованием параллельных вычислителей для серьезного счета. В ряде мощных исследовательских центрах были разработаны серьезные системы отладки правильности и эффективности. Большие усилия были предприняты для того, чтобы преодолеть влияния эффекта зонда. Этот эффект связан с тем, что внедрение в тело программы отладочных средств ведет к нарушению хода работы параллельной программы. Использовались различные (зачастую очень сложные) методы обеспечения нормальной работы отладчиков.

Интересно, что в этих («больших») системах, как правило, использовались достаточно простые виды отображения, даже, если визуализация строилась на базе появившихся в то время сред виртуальной реальности как в системе Avatar [9].

Как представляется следующий этап развития визуализации программного обеспечения параллельных вычислений, продолжающийся по сей день, можно охарактеризовать определенным падением интереса к визуальным языкам и мощным отладочным системам. Хотя работы в

этом направлении продолжаются. Примером служит сравнительно “свежая” мощная система отладки производительности Tau [10-11]. В тоже время продолжалась разработка интересных прототипов, в которых, так или иначе, решаются важные задачи визуализации, появились средства отладки для распределенных сред и для гибридных (включающих графические процессоры) вычислителей.

Схема функционирования разрабатываемых в последние годы систем отладки примерно следующая - в ходе вычислений собираются данные о работе процессов, которые являются входными при построении того или иного вида отображения, например, графов вызовов или графов потоков данных.

Отметим, что визуализация реальных параллельных программ приводит к громоздким и зачастую не интерпретируемым изображениям. Методы решения этой проблемы существуют. Например, активно используются приемы семантического зуминга, позволяющие “сворачивать” и “разворачивать” визуальные блоки, отображающие отдельные части программы. Можно использовать идеи “бесконечного экрана” и/или “полета” над визуальным пространством. Методики визуализации на базе виртуальной и расширенной реальности также могут применяться как при создании, так и при отладке параллельных программ. Однако все эти приемы, скорее, носят характер паллиативов из-за возникающих проблем с реализацией, как самого процесса вывода данных, так и с интерфейсом, удобным для программиста. Свою роль могут играть новые метафоры визуализации параллельных вычислений. Некоторые из них мы рассмотрим ниже.

3. Примеры визуализации в современных вычислительных системах

В этом разделе мы рассмотрим, как метафоры визуализации могут использоваться для представления таких понятий, связанных с параллельными и распределенными вычислениями, как трасса программы, граф вызовов и др.

Начнем наш обзор с представления трасс программ.

Трасса программы отображает динамику конкретного выполнения программы. Представление и “проигрывание” трасс программ является важным элементов отладочных систем. Обычные методы основываются на представлении программы в виде какой-либо схемы или диаграммы. Возможно также использование “прохода” (точнее “пробега”) по тексту программы, с выделением (возможно цветом) текущей позиции. В случае серьезных параллельных вычислений такие методы визуализации едва ли удовлетворяют потребностям программистов.

В свежих публикациях исследователей из Кильского университета [12-13] рассматривается использование метафор города и ландшафта для представления трасс программ. (См. описание свойств этих метафор в [14].) Интересные отображения показаны в [11], где описывается представление трассы для случая параллелизма на основе нитей. Здания представляют статичные части программного комплекса. Трасса представляется в виде лучей-нитей, протянутых между зданиями. Визуализация такой ошибки, как deadlock естественно выглядит как окрашенное пересечение этих лучей.

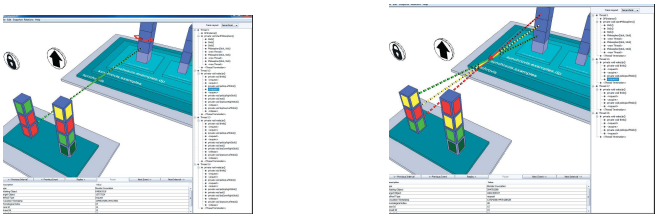

Рис. 1. Визуализация нормальной и ошибочной (deadlock) работы программы [12].

В работе [13] используются аналогичные идеи представления. Сущности программной системы (например, приложения) представляются в виде зданий, а улицы показывают потоки, связывающие эти сущности. Возможно включение/выключение режима проигрывания трассы. (Метафора города также может использоваться для представления структуры программных комплексов. См., например, [15]).

В публикациях исследователей из Потсдамского университета [16-17] для визуализации трасс сложных программных комплексов используются достаточно традиционные двумерные отображения, поддающиеся интерпретации. Резко сложнее интерпретировать виды отображения системы Zinsight [18], хотя двумерное представление потока событий в виде последовательности цветных линий также традиционно. В работах [19-20] для представления трассы используются весьма интересная метафора круговых узелков (circular bundles). Однако интерпретация графических выводов на первый взгляд представляется достаточно сложной.

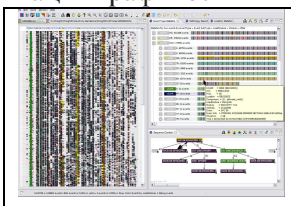	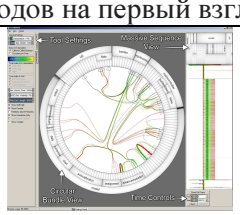	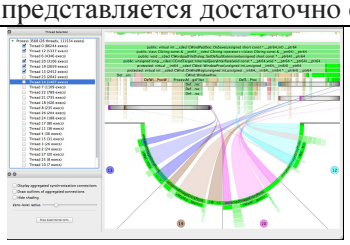	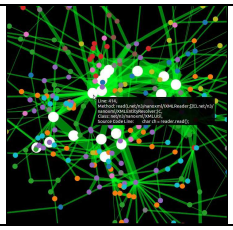
Рис. 2. Визуализация трассы в системе Zinsight [18]	Рис. 3 Визуализация трассы в виде узелков [19-20]	Рис. 4 Основное окно системы SYNCTRACE [21]	Рис. 5 Использование метафоры мозга для представления работы программы (скриншот анимации) [22]

Интересно, что исследователи из Потсдамского университета, использовавшие трехмерные метафоры города и ландшафта, построили в рамках системы SYNCTRACE весьма удачные двумерные круговые диаграммы для представления трассы многонитевых программ [21].

Следует отметить интересную метафору мозга (*"Brain" Metaphor*), использованную для анимационного представления исполнения программы в работе [22]. Идея визуализации работы мозга при предъявлении ему каких-либо стимулов перенесена на визуализацию активности программы или приложения (вызов процедур и функций, ввод/вывод и пр.). Несмотря на очень важные положения, касающиеся целей и задачи визуализации программного обеспечения, имеющиеся в [22], интерпретация полученных в рамках интересной метафоры мозга анимационных графических выводов не представляется очевидной. В работе [23] трасса большой (large) параллельной программы также описывается при помощи анимированных двумерных видов отображения.

Далее рассмотрим примеры визуализации графа вызовов. Традиционно для представления графов вызовов используются те или иные двумерные отображения. Однако, при двумерном представление графа вызовов 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. Попытки решить эту проблему в рамках двумерности представлены в [24].

Проблему нехватки места на экране дисплея, как кажется, можно решить, добавив к разрабатываемой модели еще одно измерение. В работах [25-26] (кстати, тоже написанными авторами из Потсдамского университета) используется дву-с-половиной мерная графика для представления графа вызовов. Получившийся вид отображения чем-то напоминает виды отображения, построенные на базе метафоры ландшафта.

В нашей работе [27] описаны трехмерные представления графа вызовов на базе метафоры здания (изображаются связанных между собой помещений-комнат некого здания сложной архитектуры) и метафоры молекулы.

В публикациях последнего времени можно найти немало интересных в плане визуализации решений по системам визуализации программного обеспечения. Мы приведем пример (как нам кажется перспективной) метафоры из наших недавних исследований.

Анализ параллельных программ на базе парадигмы передачи сообщений требует рассмотрения большого числа параллельно исполняемых процессов, работа которых может зависеть от событий, происходящих в том или ином процессе. Предполагается представление о времени, как об оси координат, подобной привычным трём пространственным. Время понимается как поток событий, любое изменение которого нарушает всю цепочку причинно-следственных связей. При таком подходе естественной кажется идея о возможности перемещения по времени в оба направления. Можно рассмотреть набор параллельных процессов с последовательно теку-

щими (и изменяемыми) потоками событий. Причём события-воздействия в том или ином процессе влекут реакцию, затрагивающее как процесс, в котором оно произошло, так и другие процессы. Возможно исправление ошибок за счет возвращения назад по оси времени и вмешательства в ход событий. Такой подход можно описать метафорой «машина времени». Отметим, что использование метафоры «машины времени» не требует знаний источника (научно-фантастических романов). Нами разработан прототип визуализационной составляющей системы представления параллельных процессов, которую можно будет использовать в отладочных целях. Образность при визуализации процессов – трехмерная. Процессы представляются в виде цветных цилиндров, связанных между собой тонкими «нитями». По нитям движутся шары, представляющие данные. С помощью цветов описывается состояние процесса. Пользователь может перемещаться по оси времени и менять состояние процессов. От стандартной метафоры проигрывателя метафора машины времени отличается возможностью задания событий-изменений, которые описываются “эффектом бабочки”. Этот эффект заключается в том, что казалось бы маловажное событие приводит к изменению хода процесса.

Список интересных метафор и реализованных в последнее десятилетие на их базе прототипов систем визуализации программного обеспечения параллельных вычислений, можно продолжить и дальше. Проведенные исследования и экспериментальные реализации показали значительные возможности новых метафор и построенных на их базе двумерных и трёхмерных изображений для представления сущностей параллельного программирования. В тоже время представляется, что на данном уровне реализации метафор (без использования, например, сред виртуальной реальности для проникновения “внутрь” объекта исследования) трёхмерность почти не даёт существенно нового качества при визуализации, хотя и помогает, в частности при обзоре структуры или для понимания общих положений данной проблематики.

В связи с этими рассуждениями представляется важным научиться каким-либо способом оценивать качество визуализации, проверять его пригодность для решения поставленных конкретных задач. Кроме того, возможно, не следует увлекаться поиском интересных метафор и разработкой качественной визуализации, как таковых, а перейти к созданию средств визуального сопровождения процессов разработки, отладки и анализа ПО, основываясь на изучении конкретной деятельности программистов, работающих в рамках той или иной программно-аппаратной среды.

В следующих разделах мы рассмотрим подходы к формализации и верификации визуализации и пример проектирования средств визуального сопровождения процессов разработки ПО нижнего уровня.

Рис. 6. Граф вызовов и текстовое описание кода [26]

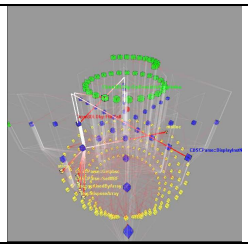

Рис 7. Граф вызовов системы рисования графа вызовов на базе метафоры здания [27]

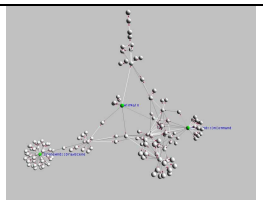

Рис. 8. Граф вызовов системы рисования графа вызовов на безе метафоры молекулы [27]

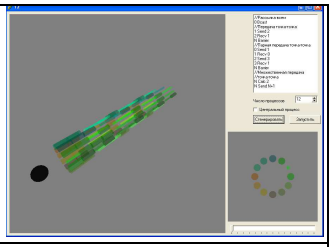

Рис. 9. Отладка MPI-программ. Метафора “машина времени” [28]

4. Верификация визуализации

Проблемы построения теории компьютерной визуализации затрагивались нами в работах [29-31] В этих работах в качестве основания теории компьютерной визуализации рассматривается семиотика. Существуют другие подходы к выбору оснований теории компьютерной визуализации. В рамках когнитивно-психологического подхода считается, что в этом качестве следует рассматривать положения, относящиеся к восприятию визуальной информации, их оценке, проблемам когниции и пр. [32-33]. Важной задачей является формализация теории визуализации. В этом плане также существуют различные подходы к формализации. Например, в [34] в качестве базы формализации предлагается использовать как семиотику, так и теорию катего-

рий. Нас интересует формализация не как абстрактный способ назвать те или иные сущности визуализации теми или иными математическими терминами, а как инструмент оценки качества и проверки правильности при выборе методик визуализации.

Верификация (проверка правильности) визуализации подразумевает наличие формальной модели, которая, по большому счету, отличается от подобных моделей для слабо формализуемых явлений, например, от формальной верификации программного обеспечения, только предметной областью. В визуализации востребована верификация таких слабо формализуемых и связанных с когнитивными аспектами подзадач, как визуализация данных большого объема, обеспечение интерактивности (задача управления), интерпретация визуализации.

Интерпретация визуализации и интерактивных манипуляций восстанавливает (или создает заново) у пользователя набор когнитивных структур, в которых представляется картина явлений. Порождение таких структур по визуальным образам и есть процесс интерпретации. Этот процесс является обратным или, точнее, двойственным визуализации.[31]. Очевидно, что любая формализация повышает уровень абстрактности, в то время как визуализация нацелена на снижение этого уровня через интерпретацию. Поэтому требование знания формальной модели от пользователя ведет к противоречию, которое можно нивелировать, применяя метафору визуализации. В данном случае метафора выступает как некий эталон верификации, в то же время, обеспечивая достаточно высокий уровень формализации. Например, метафора может пониматься как отображение исходной области (множества) на целевую область [31]. То есть, кроме формальной верификации логично рассматривать верификацию на основе метафоры или метафорическую верификацию.

Верификация, как сравнение с эталоном широко применяется в научной визуализации, например, при разработке инженерных пакетов. Трудно представить чтобы, решая, одни и те же уравнения одними и теме же методами, применяя одинаковые виды отображения, получились бы разные изображения. В принципе это возможно в результате накопления вычислительной погрешности. В работе [35] вводится понятие верифицируемой визуализации, которая отслеживает, как распространяется погрешность (неопределенность) на всем этапе вычислительного конвейера, включая визуализацию. Подобный подход в общем случае принято называть моделью с неопределенностью, а в частном визуализацией с неопределенностью, и может рассматриваться, как пример некорректной задачи по начальным данным.

Можно выделить качественные и количественные характеристики верификации или ввести две базисные функции или меры. Качественная – это степень формализации (наивная, метафорическая, формальная) и количественная, связанная с распространением неопределенности, которые уместно именовать, как полнота и точность верификации.

Одновременно с верификацией визуализации необходимо рассматривать валидацию визуализации [35]. Валидация – это мера адекватности. В математическом моделировании адекватность определяется как соответствие теории (формальной модели) практики. В принципе, формально правильная модель может быть неадекватной. Частая путаница в определение верификации и валидации объясняется достаточно просто: наивная верификация эквивалентна валидации.

Ключевые вопроса относительно неопределенности в вычислительной науке, можно сформулировать в контексте визуализации как [36]:

• Может ли быть визуализация надежным основанием для принятия решений?
• Как точность и валидация (адекватность) визуализации могут быть оценены?
• Какая уверенность или неопределенность может быть присвоена визуализации?

Рассматривая визуализацию, как решение обратной некорректной задачи, необходимо ответить на следующие вопросы:

•Существует ли решение?
•Является ли решение устойчивым?
(К этому же пункту отнесем и проверку на обусловленность или чувствительность - sensitivity);
•Является ли решение оптимальным?

Визуализацию часто рассматривают с позиции общей теории информации [37], в которой используются два термина или базисных функции: информативность и избыточность. В других работах по визуализации информативность – это наглядность, а избыточность – это вырази-

тельность, причем доказать эквивалентность этих терминов не представляется возможным из-за частого отсутствия формальных определений.

В области визуализации авторам известны три абстрактные метрики, связанные с процессом познания:

* Информационный разрыв – расстояние между тем, что мы знаем и тем, что должны знать, что бы решить поставленную задачу [38];
* Когнитивное расстояние – усилия пользователя, затраченные на интерпретацию [39];
* Инсайт (озарение, понимание) – цель визуализации, установление релевантных отношений между данными и существующей областью знаний [40] (целевой областью).

Необходимо отметить, что эти метрики являются опосредованными с точки зрения проверки адекватности.

В этой связи интересен пример систем мониторинга параллельных программ, которые проводят сбор и анализ различных метрик, совместимых с топологией числовой прямой [41].

Инструментарий таких систем предполагает, что эти метрики изначально независимы (не связаны между собой). Следовательно, пользователь всегда останется в рамках топологии графика, или другими словами для визуального анализа невозможно применить другой вид отображения, кроме вариаций графика. Таким образом, для повышения уровня инсайта, необходимо разрабатывать формальные модели как исходной, так и целевой областей, отслеживающие зависимости.

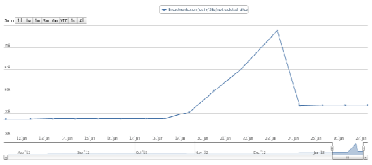

Рис. 10 Результат мониторинга атак на сайт [41].

Рассмотрим задачу робастного (с противодействием) управления, которая легко объяснима с точки зрения теории возможности (См. Рис 10). Пусть a – это уверенность, что сайт функционирует нормально, график функции должен соответствовать модели хищник-жертва с ограниченными ресурсами. Пусть b - это возможность того, что на сайт происходит атака, модель Мальтуса, экспоненциальный рост графика. Реальное значение количества IP-адресов всегда будет в интервале [a,b] Фактически нас интересуют не сами графики, а скорость сближения графиков a/b(t) или эффективность контрмер. В общем случае это отношение можно рассматривать, как темпоральное нечеткое число. Также необходимо отметить, что у контрмер достаточно высокая латентность или плохая обусловленность системы, связанная с отсутствием дифференцируемости или с не возможностью отследить скорость изменение графика на достаточно малом промежутке времени. Атакующий также может проанализировать график и изменить стратегию атаки, что приведет к дальнейшему росту латентности. В этом смысле, идея создания клона сайта кажется интересной, фактически атакующий будет наблюдать экспоненциальный рост, а реальная работа будет проходить в нормальном режиме.

В сфере визуализации хорошо зарекомендовало себя применение комплексного или множественного вида отображения. Это понятие подразумевает несколько разделенных видов отображения в процессе интерпретации и взаимодействия, между которыми устанавливаются взаимосвязи. В качестве примера такого вида отображения можно привести аналогию из области черчения, когда трехмерное тело представляется виде трех проекций. Очевидно, что для рассмотрения и формализации данного примера можно ввести несколько групп базисных функций, таких как Информативность и избыточность; Полнота и точность, обеспечивающие целостное восприятие и детализацию.

Результатом (целью) применения множественного вида отображения может являться не только сокращение объема информации, но и понижение размерности, в том числе и когнитивной. В этом случае множественный вид отображения удобно рассматривать, как произведение бикомпактов, обеспечивающие наследование свойств (произведение бикомпактов - бикомпакт).

Как пример явного разложения по двум базисным функциям можно привести множественный (бинарный) вид отображения, включающий в себя миникарту и основной вид отображения, для которого можно поставить задачу минимакса: с одной стороны, обеспечение целостно-

го восприятия, но не точно, с другой – детально (более точно), но не всей информации (применение уровня детализации).

5. Визуальное сопровождение процесса разработки программного обеспечения

В качестве примера разработки средств визуального сопровождения процесса разработки мы рассмотрим визуализацию графов, соответствующих программам на языке ассемблера отечественных процессоров семейства Мультиклет с целью верификации и оптимизации программ.

Архитектура процессоров семейства Мультиклет обеспечивает простой способ внеочередного исполнения команд, явный контроль программиста за упорядочиванием памяти (memory ordering), прозрачную для приложений реконфигурацию во время исполнения, отказоустойчивость - в том числе и благодаря оригинальному способу кодирования машинных команд [42]. В основе этого кодирования лежит идея пересылать данные между элементарными процессами исполнения инструкций не через регистры (как это сделано в традиционных архитектурах) или через очереди сообщений (как это предполагается в архитектурах семейства EDGE), а напрямую, от процесса к процессу (здесь мы под процессами понимаем конструкции в большой степени соответствующие теории CSP, предложенной Хоаром). Данная концепция напрямую выражена в машинном языке процессоров Мультиклет. В этом языке инструкции задают три типа элементарных процессов: (1) процессы, приводящие к записи в память и в регистровый файл или к передаче управления на некоторый блок кода; (2) процессы, приводящие к чтению из памяти или из регистрового файла; (3) прочие процессы, в основном, формирующие результаты логических и арифметических вычислений. Каждый процесс из группы (2) или (3) формирует сообщение с результатом своей работы, которое можно идентифицировать по положению задающей его инструкции в участке кода, называемом параграфом. Соответственно, каждый процесс из групп (1) или (3) нуждается в исходных данных, и то, какие сообщения следует принять этому процессу для продолжения работы указывается в инструкции, которая этот процесс формирует. Эти указания даны в виде описания позиций в блоке кода тех инструкций, процессы выполнения которых приведут к формированию нужных результатов.

Проиллюстрируем это на примере цепочки инструкций ***“rdl 0xabc; rdl 0xabc; addl @1, @2; wrl @1, 0xabc”***. Исполнение инструкции *addl*, стоящей на 2-ой позиции (счёт с 0) приведёт к запуску процесса, который для своего продолжения должен получить результаты исполнения 0-ой (задаётся операндом *@1*) и 1-ой (задаётся операндом *@2*) инструкций. Процесс исполнения *wr*l на позиции 3 будет ожидать сообщения с результатом инструкции *addl* на позиции 2. Заметим, что процессоры семейства Мультиклет не задают никакой явной модели упорядочивания памяти. Все четыре элементарных вычислительных процесса в цепочке могут (и будут) запущены параллельно. Но результатом выполнения процесса всего описанного процесса будет ожидаемая запись удвоенного значения из ячейки 0xabc в неё же.

Такой метод представления и кодирования вычислений, с одной стороны, даёт процессорам Мультиклет их технические преимущества, но, с другой, создаёт определённые сложности при анализе программ, составленных автоматически (компилятором) или же вручную на языке ассемблера для машин этого семейства.

Основную сложность представляет то, что программист больше не может при неинструментальном анализе программы связать некоторую синтаксическую конструкцию (для традиционных ассемблеров - имя регистра) с неким значением. В разных участках параграфа ссылка *@7* будет указывать на разные инструкции, соответственно, и на результаты разных в общем случае элементарных вычислительных процессов.

Для упрощения анализа исходных текстов на ассемблере процессоров Мультиклет и было предложено визуализировать графы различных участков программы. В отличии от традиционных ассемблеров, в случае с Мультиклетами, задача построения такого графа проста: фактически нужно перевести естественным образом код программы в описание графа, например, на языке Dot (система GraphViz – [43]). Что и было проделано.

Изначальные результаты такого перевода оказались недостаточно выразительными, так как этот перевод осуществлялся без учёта особенностей алгоритмов размещения вершин, используемых в GraphViz для визуализации. Постепенно многие из этих особенностей удалось учесть и получить приемлемые визуализации участков кода, по которым можно было проследить интересующий нас, как программистов, поток данных в участке кода.

Дополнительной особенностью результатов такой визуализации оказалась возможность узнавать некоторые алгоритмы по их графическому изображению. Например, при визуализации тела цикла классического алгоритма FFT можно наблюдать подобие классической “бабочки”.

Важным для практики результатом работы является возможность быстро прослеживать степень возможного параллелизма анализируемого участка кода, к которой процессоры семейства Мультиклет чувствительны. Код с небольшой степенью параллелизма естественным образом визуализируется как вытянутая нить операций. Такое прослеживание необходимо при отладке производительности.

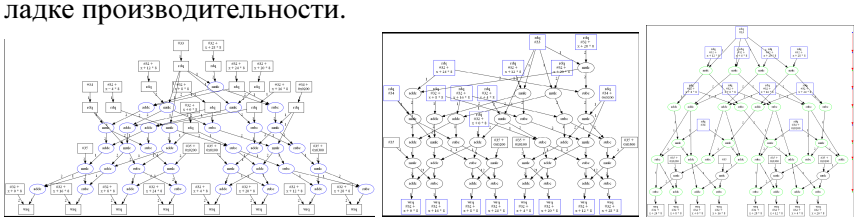

Рис. 11 – а) пример начального результата, б) уменьшение количества узлов, в) улучшение структуры.

Далее мы рассмотрим возможности визуального сопровождения, используемые в процессе разработки компилятора с языка Си-99.

В процессе работы транслятор с языка программирования высокого уровня на язык ассемблера строит в памяти структуры данных, насыщенные перекрёстными ссылками. Нисколько не преувеличивая, можно сказать, что транслятор с практически используемого языка строит структуры перенасыщенные такими перекрёстными ссылками. Перенасыщенные в том смысле, что без дополнительных инструментов ни пользователь, ни разработчик компилятора не способен достаточно точно восстановить задаваемую ими семантику. Из-за этого теряется смысл, допустим, традиционного подхода к отладке программ, в котором отладчик графической IDE позволяет программисту просматривать значения в определённом небольшом (ограниченном возможностями человека) наборе переменных. Семантика, сохранённая в сложных структурах данных транслятора по такому небольшому набору значений не восстанавливается. Соответственно, таким методом отыскать логические ошибки зачастую оказывается невозможным. Следовательно, в разработке компилятора необходимо прибегать к другим инструментам отладки. Традиционно для этого большими коллективами разработчиков используются внутренние механизмы отладки и проверки корректности работы транслятора. Но для организации такой внутренней системы требуется затрачивать время и ресурсы на формулирование формальных критериев корректности согласованности таблиц, на формулирование запросов об их структуре, на разработку алгоритмов, которые воплощают эти формулы, и на программирование самих алгоритмов. Большие коллективы обладают ресурсами для решения этих задач. Небольшим коллективам, работающим в условиях жёсткой конкуренции и, как следствие, в условиях постоянной нехватки времени и ресурсов, такой метод не подходит. Ситуация усугубляется ещё и тем, что сами внутренние механизмы отладки являются сложным кодом и редко обходятся без уже своих ошибок, которые также нужно отыскивать и исправлять. Всё это приводит к необходимости отыскать альтернативное решение.

Своё решение мы основываем на следующем факте: хотя семантические таблицы, выстраиваемые в процессе трансляции, и перенасыщены перекрёстными ссылками, сами эти таблицы имеют регулярную структуру, и их достаточно просто распечатать относительно простыми процедурами. Такие распечатки таблиц, естественно, не поддаются анализу без дополнительных инструментов: они слишком большие и всё также перенасыщены перекрёстными ссылками, пускай теперь и чисто в текстовом виде.

Но теперь речь идёт о тексте с перекрёстными ссылками, и для его анализа теперь мы можем использовать арсенал инструментов и опыт по работе с HTML, такими как динамическая

подсветка текста, навигация по ссылкам, свёртка и развёртка сложных текстовых структур. Переведя простое текстовое представление семантических таблиц компилятора в HTML-документ, оснащённый дополнительным кодом на JavaScript для анимации (в основном, изменениями цвета) семантических таблиц и навигации по ним, мы существенно упрощаем себе задачу по анализу корректности работы транслятора. Преимуществом подхода является то, что при помощи основных базовых операций: подсветка мест упоминания интересных ссылок, прямой (от ссылки к объекту) и обратной (от объекта к ссылкам) навигации по тексту - программист получает возможность в ручном режиме выполнять произвольные, иногда даже плохо сформулированные, запросы к структуре семантических таблиц. Предложенный достаточно простой инструмент отладки на основе анимации текста подсветкой и навигацией отлично зарекомендовал себя в поиске ошибок, сложно локализуемых другими способами.

6. Заключение

В нашем исследовании рассматривались методы представления информации в средах визуализации ПО параллельных вычислений. В упрощенном виде ситуация представляется следующей – по сравнению с 90-ыми годами очень мало мощных визуальных сред отладки; основное внимание сосредоточено на создании прототипов, на основе новых метафор визуализации и эффектных видов отображения. Возникло впечатление уменьшения серьезного интереса к разработке систем визуализации ПО. Визуализация ПО (в том числе и для активно развивающихся параллельных вычислений) стала казаться чем-то необязательным. В то же время разработчики программного обеспечения параллельных вычислений, в том числе программного обеспечения низкого уровня, работающие с элементами процессоров сложной архитектуры, ставят задачи представления объектов их интереса в интерпретируемом виде.

Анализ литературы и наш собственный опыт показывает, что оправдано использование в реальных системах достаточно простых, но четко интерпретируемых методов представления. Такие методики могут показаться примитивными, особенно по сравнению с цветными анимациями, разработанными на базе новых метафор. И трехмерность, и анимация могут быть очень полезными, но они должны применяться в рамках использованных систем программирования, а не требовать перехода в некоторый новый и непонятный для программиста мир визуализации (да еще и созданный на базе средств виртуальной реальности с возможностями гиперпогружения и киберболезни). Возможно, не следует увлекаться поиском интересных метафор и разработкой качественной визуализации, как таковых, а перейти к созданию средств визуального сопровождения процессов разработки, отладки и анализа ПО, основываясь на изучении конкретной деятельности программистов, работающих в рамках той или иной программно-аппаратной среды. (См. также идеи по использованию ментальной образности программистов при разработке средств визуализации ПО в интересной работе [44].)

В связи с идеями визуального сопровождения работы программиста будут решаться задачи визуального моделирования программ для новых процессоров и визуализации отладки программ нижнего уровня. В этом случае будут использоваться различные методики представления программных сущностей, в том числе трехмерные и анимационные виды отображения.

Литература

1. Авербух В.Л., Байдалин А.Ю., Разработка средств визуализации программного обеспечения параллельных вычислений. Визуальное программирование и визуальная отладка параллельных программ. // ВАНТ. Сер. Математическое моделирование физических процессов, 2003, вып. 4. Стр. 68-80.

2. Авербух В.Л., Байдалин А.Ю. Разработка средств визуализации программного обеспечения параллельных вычислений. Оптимизация параллельных программ // ВАНТ, сер. Математическое моделирование физических процессов, вып. 1. Стр. 70-79, 2004.

3. Авербух В.Л., Байдалин А.Ю., Исмагилов Д.Р., Казанцев А.Ю., Состояние дел в визуализации программного обеспечения параллельных вычислений // ГРАФИКОН-2005. Труды Конференции. Новосибирск. ИВМиМГ СО РАН. Стр. 179-186.

4. Посыпкин М.А., Соколов А.А. Обзор методов автоматизации мониторинга, анализа и визуализации поведения параллельных процессов, взаимодействующих с помощью передачи сообщений. Препринты Института системного программирования РАН, Препринт 7, 2005 г.
5. Averbukh V., Bakhterev M. The analysis of visual parallel programming languages // ACSIJ. Advances in Computer Science: an International Journal, Vol. 2, Issue 3, No. 4, July 2013. Pp. 126-131.
6. Socha D., Bailey M.L., Notkin D. Voyeur: Graphical Views of Parallel Programs //Proceeding of the ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging. May 5-6 1988, Wisconsin. SIGPLAN Notices. Vol.24. No 1. (Janiary 1989) pp. 206-215.
7. Beguelin A., Nutt G. Visual parallel programming and determinacy: a language specification, an analysis technique, and a programming tool // Journal of Parallel and Distributed Computing. Vol. 22, Issue 2, Aug. 1994. Pp. 235-250.
8. Abramson D., Foster I., Michalakes J., Sosic R. Relative Debugger: A New Methodology for debugging Scientific Applications // Communication of the ACM. V. 39, No. 11 (1996). Pp. 69-77.
9. Reed D., Scullin W., Tavera L., Shields K., Elford Ch. Virtual Reality and Parallel Systems Performance Analysis //IEEE Computer, V.28, N 11, (1995). Pp. 57-67.
10. Shende S.S., Malony A.D. The Tau Parallel Performance System // International Journal of High Performance Computing Applications. Vol. 20. Issue 2, May 2006 Pp. 287-311.
11. Malony A., Shende S., Spear W., Chee Wai Lee, Biersdorff S. Advances in the TAU Performance System // Proceedings of the 5th International Workshop on Parallel Tools for High Performance Computing, September 2011, ZIH, Dresden. Springer Berlin Heidelberg. 2012. Pp. 119-130.
12. Waller J., Wulf Ch., Fittkau F., Dohring Ph., Hasselbring W. SynchroVis: 3D Visualization of Monitoring Traces in the City Metaphor for Analyzing Concurrency // First IEEE Working Conference on Software Visualization (VISSOFT), 2013. 4 pp.
13. Fittkau F., Waller J., Wulf Ch., Hasselbring W. Live trace visualization for comprehending large software landscapes: The ExplorViz approach // Proceedings of the 1st Working Conference on Software Visualization (VISSOFT), IEEE Computer Society, 2013. 4 pp.
14. Авербух В.Л., Исмагилов Д.Р. Свойства метафор визуализации и выбор методов представления данных о функционировании программных комплексов // ПаВТ'2009. Труды международной научной конференции. Челябинск. ЮУрГУ, 2009. Стр. 343-349.
15. Kobayashi K., Kamimura M., Yano K., Kato K., Matsuo A. SArF Map: Visualizing Software Architecture from Feature and Layer Viewpoints // 21st IEEE International Conference on Program Comprehension (ICPC), 2013. Pp. 43 – 52.
16. Trumper J., Bohnet J., Dollner J. Understanding Complex Multithreaded Software Systems by Using Trace Visualization // Proceedings of the 5th International Symposium on Software Visualization, ACM. 2010. Pp.133-142.
17. Trumper J., Telea A., Dollner J. Viewfusion: Correlating structure and activity views for execution traces // Proc. of 10th Theory and Practice of Comp. Graph. Conf. Euro. Asso. for Comp. Graph., 2012, pp. 45–52.
18. De Pauw W., Heisig S. Zinsight: a visual and analytic environment for exploring large event traces // Proceedings of the 5th international symposium on Software visualization, ACM, 2010. Pp. 143-152.
19. Cornelissen B., Holten D., Zaidman A., Moonen L., van Wijk J.J., van Deursen A. Understanding execution traces using massive sequence and circular bundle views // Proc. of the 15th IEEE Int. Conf. on Program Comprehension. IEEE, 2007, pp. 49–58.
20. Cornelissen B., Zaidman A., Holten D., Moonen L., van Deursen A., van Wijk J.J. Execution Trace Analysis through Massive Sequence and Circular Bundle Views // Journal of Systems and Software. Vol. 81. 2008. Issue 12, December. Pp. 2252-2268.
21. Karran B., Trumper J., Dollner J. SYNCTRACE: Visual Thread-Interplay Analysis // Proceedings of the 1st Working Conference on Software Visualization, IEEE Computer Society, 2013. 10 pp.
22. Palepu V.K., Jones J.A. Visualizing constituent behaviors within executions // Proceedings of the 1st Working Conference on Software Visualization (VISSOFT), IEEE Computer Society, 2013. 4 pp.
23. Sigovan C., Muelder Ch.W., Kwan-Liu Ma Visualizing Large-scale Parallel Communication Traces Using a Particle Animation Technique // Computer Graphics Forum. Vol.32. June 2013. Pp. 141-150.

24. LaToza T., Myers B. Visualizing call graphs // 2011 IEEE Symp. on Visual Languages and Human-Centric Computing, 2011, pp. 117–124.
25. Bohnet J., Dollner J. Visual exploration of function call graphs for feature location in complex software systems // Proc. of Symp. Soft. Vis., 2006, pp. 95–104.
26. Bohnet J., Dollner J. Facilitating Exploration of Unfamiliar Source Code by Providing 21/2D Visualizations of Dynamic Call Graphs // Proceeding of 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. Pp. 63-66.
27. Авербух В.Л., Байдалин А.Ю., Исмагилов Д.Р., Казанцев А.Ю. Трёхмерные методики визуализации программного обеспечения параллельных и распределённых вычислений // Труды ПаВТ'2008. Челябинск, Издательство ЮУрГУ, 2008. Стр. 283-288.
28. Авербух В.Л., Михайлов И.О., П.В. Небогатикова П.В., Поиск новых подходов к визуализации процессов // XIV Международная конференция «Супервычисления и Математическое Моделирование». Тезисы. ФГУП «РФЯЦ ВНИИЭФ». Саров. 2012, стр. 15-17.
29. Авербух В.Л., К теории компьютерной визуализации // Вычислительные технологии. Т. 10, N 4, 2005, стр. 21-51.
30. Averbukh V. Theory as a Bridge between Education, Research and Development in Human-Computer Interaction and Computer Visualization // Journal of International Scientific Publication: Educational Alternatives. 2012. Vol. 10. Part 3. Pp.151-158.
31. Авербух В. Семиотический подход к формированию теории компьютерной визуализации // Научная визуализация, 2013. Квартал: 1. Том: 5. Номер: 1. Стр. 1-25.
32. Workshop at Visweek 2010: The Role of Theory in Information Visualization, October 25, 2010, Salt Lake City, Utah, USA. http://eagereyes.org/infovis-theory-workshop
33. Liu Zhicheng, Stasko J.T. Mental Models, Visual Reasoning and Interaction in Information Visualization: A Top-down Perspective // IEEE Transactions on Visualization and Computer Graphics, Vol. 16, No. 6, November/December 2010, pp. 999-1008.
34. Vickers P., Faith J. Rossiter N. Understanding Visualization: A Formal Approach Using Category Theory and Semiotics // IEEE Transactions on Visualization and Computer Graphics. Vol. 19. No. 6. June, 2013. Pp. 1048-1061.
35. Kirby R., Silva C. The need for verifiable visualization // IEEE Computer Graphics and Applications. Sep 2008. Vol.28. No5. Pp.78 –83,
36. Fout N., Ma K.-l. Reliable Visualization: Verification of Visualization based on Uncertainty Analysis/ Tech. rep., University of California, Davis, 2012
37. Dasgupta A., Min Chen, Kosara R. Conceptualizing Visual Uncertainty in Parallel Coordinates // Comput. Graph. Forum 31(3) (2012). Pp. 1015-1024.
38. Info-gap decision theory: http://en.wikipedia.org/wiki/Info-gap_decision_theory
39. Jacob R.J.K. Direct Manipulation // Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, (Atlanta, GA, 1986), N.Y. 1986. V.1, pp. 384-389.
40. North Ch. Toward Measuring Visualization Insight // IEEE Computer Graphics and Applications May/June 2006, Volume: 26, Issue: 3, pp. 20-23.
41. Васёв П.А., Согомонян М.С. Веб-система визуализации, анализа и мониторинга работы программ // ПАВТ-2013., ЮжУрГУ. Челябинск. Сборник Трудов. Том 2. С. 586.
42. Стрельцов Н.В. Архитектура и реализация мультиклеточных процессоров // Труды V Международной научной конференции «Параллельные вычисления и задачи управления» - Москва, 26-28 октября 2010. С. 1087-1104.
43. Graphviz - Graph Visualization Software (http://www.graphviz.org/)
44. Petre M. Mental imagery and software visualization in high-performance software development teams // Journal of Visual Languages and Computing, 21 (3), 2010. Pp. 171–183.

Работа выполнена при поддержке Программы фундаментальных исследований УрО РАН "Информационные, управляющие и интеллектуальные технологии и системы", проект 12-П-1-1034.