Анализ средств визуального программирования параллельных вычислений*

В.Л. Авербух[1], М.О. Бахтерев[1]

ИММ УрО РАН, г. Екатеринбург[1]

В статье рассматривается история развития и потенциал использования средств визуального программирования параллельных вычислений. Анализ систем, разработанных, начиная с 80-ых годов прошлого века, позволяет выявить стоящие перед разработчиками проблемы и предложить некоторые методы их решения.

1. Введение

Наметившаяся в 70-ых годах XX века массовая компьютеризация потребовала резкого увеличения объема программного продукта, роста производительности труда программистов и, конечно, роста числа программистов. С появившимися в эти годы методиками визуального программирования связывались значительные надежды на облегчение обучения, увеличение производительности труда программистов, повышение надежности программного продукта и уменьшение сложности процесса программирования. Еще большие надежды возлагались на эти методики в связи с параллельным программированием, которое заведомо считается более сложным, чем традиционное последовательное. Работы в области визуального программирования параллельных вычислений начались, практически, одновременно с созданием первых сред визуального программирования последовательных программ. В 90-ых годах отмечался особенно большой интерес к данному направлению. Исследования и опытные разработки продолжались и в последующие годы. Мы рассматривали состояние дел в данной области в работе [1], опубликованной в 2003 году. В этой работе мы еще раз проанализируем итоги развития визуального программирования параллельных вычислений за почти четыре десятилетия, что поможет выявить перспективы его развития и возможности создания эффективных сред проектирования и разработки.

2. Визуальное программирование

Термин “Визуальное программирование” подразумевает использование визуальных выражений (графов, схем, диаграмм, пиктограмм и т.п.) в процессе программирования, что дает возможность специфицировать программу в двумерном или трехмерном виде.

Традиционно визуальные языки программирования подразделяются на иконические языки, использующие иконы (пиктограммы) для представления объектов, операций и функций, и диаграмматические языки, основанные на использовании схем и диаграмм, так или иначе описывающие программные процессы. В отдельный класс выделялись языки, ориентированные на использование таблиц, формуляров и бланков.

Как уже указывалось, при появлении визуального программирования в 80-ых годах на него возлагались большие надежды в плане упрощения процесса программирования. Считалось, что визуальный способ описания программы более доступен мышлению начинающего программиста, так как картинка отражает реальный мир, тогда как текст лишь указывает на объекты реального мира. Кроме того, многомерность графики может увеличить информативность по сравнению с одномерным потоком текста за счет использования, например, формы, размера, цвета, текстуры, направления или расстояния. То есть визуальный метод описания был призван понизить уровень абстракции представления алгоритмов. Важным представлялось то, что возникала возможность перейти при программировании от мышления в рамках виртуальной (Фор-

* Работа выполнена при поддержке программы Президиума РАН № 18 "Алгоритмы и математическое обеспечение для вычислительных систем сверхвысокой производительности", а также проекта 12-П-1-1034 УрО РАН.

тран, Алгол, etc.) машины к мышлению более обозримыми и компактными визуальными образами. Предполагалось активное использование визуальных метафор на базе какой-либо естественной образности, а также схем различного типа (конечных автоматов, графов потоков данных, диаграмм состояний, сетей Петри).

Первые реализации сред визуального программирования были основаны на так называемой *исполняемой графике*, когда имеется возможность описывать алгоритмы в визуальной форме и исполнять визуальные программы непосредственно на компьютере без перевода в текст на «нормальном» языке. Отсюда вытекает идея о том, что визуальные языки должны иметь графические представления для всех элементов программы, как для управляющих конструкций, так и для структур данных. Казалось, что за счет использования визуальных методик проектирования программ (например, диаграмм Нэсси-Шнейдермана) можно будет вообще отказаться от этапа кодирования. Так как визуальное представление дает возможность анимировать (оживить) изображения соответствующих программных конструкций, то с помощью анимационной графики предполагалось разрешить проблемы, зачастую возникавшие у начинающих программистов, когда они не могли соотнести статичный текст программы и тот процесс, который этим текстом генерировался.

Значительная часть реализаций сред визуального программирования в 80-ых и 90-ых годах базировалась на использование диаграмматических языков. Диаграмматические языки характеризуются строго определенным, формализованным словарем, состоящим из сравнительно небольшого количества элементов. Также, как правило, строго определен пространственный синтаксис - расположение элементов схем в пространстве и относительно друг друга. Многие системы проводили размещение элементов схем автоматически. Работа с системами программирования на базе диаграмматических языков, как правило, идет по общей схеме - пользователь, применяя систему меню, указывает на графические шаблоны - элементы диаграмм и размещает их в рабочем поле. После этого происходит заполнение соответствующих текстовых полей шаблонов. Системы программирования на базе диаграмматических языков по существу используют гибридные - текстово-графические языки.

Существует много примеров использования в качестве базовой метафоры блок-схем и диаграмм Несси-Шнейдермана. Эти диаграммы базируются на представлении потока управления программ. Были широко распространены визуальные языки, описывающие в том или ином виде потоки данных. Простая структура графов облегчает модульное проектирование и дает возможность применять схемы на всех уровнях описания проектируемых систем. Нет необходимости применять специальные языковые средства увязки отдельных модулей. Примеры визуальных языков на базе конечных автоматов, сетей Петри, HIPO-диаграмм встречались реже. В 90-ые годы, на следующем этапе развития визуальных языков, появились реализации на базе UML диаграмм.

В диаграмматических системах, основанных на представлении потока управления, обычно отсутствуют средства графического представления данных программы. Все, что касается данных должно вводиться в обычном текстовом виде, и все эти системы нуждаются в обширном объеме текстового ввода. Многие системы требуют на каком-то этапе детализировать фрагменты программы, для чего необходимо делать вставки на обычном текстовом языке программирования. Использование графов потоков данных влечет аналогичные проблемы - отсутствие высокоуровневых управляющих структур, приводящее к усложнению и запутыванию диаграмм, отсутствие средств поддержки нетривиальных структур данных и т.д. Кроме этого при применении графов потоков данных единственный способ передачи семантической информации - это использование имен узлов и дуг. У систем, основанных на других схемных метафорах обычно сильно ограничена область применения. Для разрешения проблем, возникающих при использовании диаграмм потоков данных и потоков управления, применялись смешанные диаграмматические метафоры, объединяющие, например, диаграммы потоков данных и сети Петри. Но имели место лишь единичные примеры таких решений.

Реализации систем на базе иконических языков часто базируются на расширенных моделях потоков данных. В этом случае в узлах графа помещается картинка, представляющая, как правило, заранее написанную функцию обработки данных. Существуют интересные примеры иконических языков с использованием естественной образности, достаточно точно отображающей смысл той или иной функции.

Графы потоков данных, также как и графы потоков управления, достаточно легко анимируются. Однако систем исполняемой графики с анимацией немного. Существуют примеры разработки в рамках идей “исполняемой графики” визуальных вариантов “обычных” языков, включая языки функционального программирования, для которых не просто подобрать адекватные методики визуализации основных понятий.

Диаграмматические языки оказались наиболее востребованным типом визуальных языков программирования. Именно диаграмматическими языками на базе потоков данных являются визуальные языки, встроенные в ряд математических пакетов. Сравнительно свежим примером классического диаграмматического языка на базе потоков данных является Microsoft Visual Programming Language (VPL). Можно вспомнить, что большую роль играют визуальные средства на базе диаграмматической образности в таких инструментальных средах программирования как Visual BASIC, Delphi и пр. Правда, в данном случае под визуальным программированием понимается не совсем то (или совсем не то), что первоначально подразумевалось в 80-ых годах. Теперь это не системы исполняемой графики, которые казались целью развития визуальных программных сред в 80-ых годах. В известных средах визуального программирования основную роль играет не визуальное задание структуры потока управления или потока данных, а задание структуры интерактивного поведения прикладной программы. Опыт создания иконических языков программирования сыграл свою роль при проектировании систем графического (иконического) интерфейса.

3. Визуальные языки параллельного программирования

Первые реализации визуальных языков параллельного программирования появились, практически, сразу же после появления последовательных аналогов в 80-ые годы и даже чуть ранее.

Попытки визуального задания параллелизма на первых порах ограничивались тем, что участки визуального представления программы (графа потока данных или схемы, отображающей графы других типов), которые, по мнению программиста, можно распараллелить, выделялись каким-либо способом (обычно двойной или жирной линией). В ранних средах такого типа не было явной визуальной поддержки сущностей параллельного программирования, связанных с посылками сообщений между процессами или их порождения/уничтожения. На следующем этапе кроме подобных подсказок компилятору в языки вводились описания достаточно сложных конструкций параллелизма. В визуальных языках параллельного программирования при задании последовательной части программы использовались традиционные диаграмматические методики описания потока управления. В ряде языков использовались наборы простых пиктограмм, представляющих различные программные конструкции того или иного языка программирования. (Например, VISO - визуальный вариант языка Occam [2].)

В ходе развития визуальных параллельных языков программирования выявилась также тенденция к созданию “концептуальных систем”, когда разработчиками выдвигалась определенная концепция описания параллелизма и соответственно параллельного программирования, для которых подбиралась своя методика визуализации. Была поставлена задача разработки систем визуального программирования, в которых весь цикл разработки (непосредственное программирование, отладка правильности и настройка производительности) будет вестись в рамках одной ментальной модели. Можно даже сказать, что при проектировании подобных сред визуального программирования важным является формирование самой ментальной модели функционирования параллельных программ. Другой вопрос - решены ли эти задачи и даже решаемы ли они в принципе.

Методики непосредственного создания программ и визуального представления в подобных средах, несмотря на использование различных подходов к заданию параллелизма, похожи. Программист создает общую схему параллельной (или распределенной) программы, а затем конкретизирует ее путем задания конкретных деталей на отдельных участках. Образность почти всех систем - диаграмматическая. Набор графических эле-

ментов традиционно невелик и, в целом, упрощен, даже, если и используются какие-либо пиктограммы. Анимация для описания динамики процессов не используется.

Среди ранних “концептуальных” решений обратим внимание на появившиеся в начале 90-ых годов языки CODE [3] и Phred [4].

На языке CODE программа представляла собой граф потоков данных решения данной задачи, состоящий из узлов-процессов (разноцветных окружностей с именем), в которых проводятся вычисления и дуг (стандартных стрелок), соединяющих порты этих узлов. На следующем этапе узлы аннотируются (описываются в текстовом виде) порты ввода и вывода, а также задаются достаточно сложные правила запуска узлов, содержащие условия, по которым будут работать данные узлы.

При проектировании языка Phred особое внимание уделялось проблеме *детерминированности* в параллельном программировании. Недетерминированные программы могут содержать трудно обнаруживаемые ошибки. Если система программирования может предупреждать программиста о потенциальной недетерминированности, программист имеет больше шансов избежать западни, связанной с “гонкой сообщений” параллельной программы. Язык Phred рассматривается в связи с проблемами восприятия параллелизма вычислений. Его называют *координационным языком*, в противоположность обычным языкам, описывающим только вычисления.

Программа на языке Phred состоит из *потока управления*, *потока данных* и *множества интерпретаций узлов* (то есть процедур, запускаемых при достижении узла). Визуальная компонента языка Phred обеспечивает изображение потоков управления и данных, создаваемых непосредственно при помощи графического редактора. Интерпретации узлов представляются спецификациями процедур. Дуги в графе потока управления представляют дизъюнктивные и конъюнктивные потоки управления. Таким образом, граф потока управления языка Phred обеспечивает представление последовательных потоков, переключение управления, параллелизм и синхронизацию.

Важным примером концепций описания параллелизма может служить язык Visper [5]. Существенным в проекте является введенное понятие графа взаимодействия процессов - *Process communication graph*. (Это понятие, включая язык PCG, используется также в ряде родственных разработок визуальных языков.) Граф взаимодействия процессов комбинирует графы потока управления и потока данных в единый визуальный формализм. Насколько можно понять из публикаций, в рамках многолетних исследований и разработок рассматривались различные идеи по визуализации процесса параллельного программирования. Поэтому какого-то канонического варианта язык не существует. По нашему мнению важной идеей в Visper'e является идея использования методик отображения параллельности, которые ранее использовались на этапах отладки и настройки эффективности параллельной программы. В основу этих методик положены (несколько модернизированные) понятия *карты параллельности* и *диаграммы пространства-времени.*

Карта параллельности отображает возможную параллельность задачи. Это одновременно структура данных для перезапуска (переигровки) потока управления и графический метод представления параллельных процессов. Карта показывает историю проработки процессов в виде потока событий на временной шкале.

Динамика выполнения параллельной программы в диаграммах пространства-времени представляется как поток событий в двумерном пространстве, где одна ось показывает время, а вторая - реальные отдельные процессоры. При модернизации этих понятий в рамках Visper'a временная ось переосмысливается как ось потока управления, а процессорная ось расширяется за счет введения понятия группы. Граф взаимодействия процессов определяется на трехмерном пространстве программирования. По оси ***X*** показываются процессоры и группы процессоров, а также потоки данных, которыми они обмениваются. Ось ***Y*** определяет поток управления программы, а на оси ***Z*** показано количество процессоров.

Существует еще целый ряд подходов к созданию визуальных параллельных языков, реализованных уже в первое десятилетие XXI века.

В языке Vorlon [6] реализована модель параллельного исполнения потока объектов (*parallel object-flow execution model*). Эта модель объединяет объектно-ориентированную модель и модель потока данных. Таким образом, существует возможность обеспечить как задание паралле-

лизма, так и всей сложности конкретной прикладной задачи. При этом такие аспекты параллелизма, как синхронизация, обеспечиваются за счет конструкций, поддерживающих поток данных, а задание типов и их взаимосвязи дает возможность описывать все многообразие прикладной области.

Параллельный граф потока объектов состоит из узлов и дуг. Узлы представляют некоторую форму вычисляемого элемента, а дуги описывают взаимозависимости потоков управления и возможные маршруты передачи параметров между узлами.

Также с идеями потока данных и объектно-ориентированного программирования связан интересный проект языка HiPPO (High Performance Parallel Object-oriented) [7]. Рассматривается модернизация графа потока данных, который в данном случае является потоком ссылок на объекты.

VisualGOP [8], еще одна реализация среды параллельного программирования, была осуществлена с использованием графово-ориентированной модели программирования (*graph-oriented programming model*). Основа среды - язык GOP, на котором путем создания логического графа описывается логическая структура параллельных или распределенных программ. Этот граф служит для представления коммуникативных связей и синхронизации между отдельными программами в распределенной среде выполнения. VisualGOP не зависит ни от конкретных языков программирования, на которых написаны эти программы, ни от платформ, на которых программы функционируют.

Важным и активно разрабатываемым направлением в визуальных языках параллельных вычислений являются диаграмматические языки, базирующиеся на парадигме посылки сообщений между процессами. Работа в средах программирования, разработанных на базе этих языков, во многом напоминала работу в других средах визуального программирования - пошаговая детализация программы, представленной в визуальном виде; использование гибридного подхода - и графика и текстовые описания.

Наибольший интерес в этой связи представляет по нашему мнению язык Grapnel [9], в котором достаточно полно (на момент его создания в середине 90-ых годов) отражены достижения, как в параллельном, так и в визуальном программировании. В частности в языке Grapnel реализована визуальная поддержка механизма динамического порождения и уничтожения процессов и возможность использования предопределенных шаблонов топологий процессов.

При решении многих прикладных задач процессы размещаются на базе определенных топологий, таких как линейный массив, кольцо, сетка, дерево, etc. Механизм использования стандартных топологий служит эффективным средством создания динамически расширяемых распределенных программ, то есть таких программ, размерность топологии которых становится известно только во время работы программ. Тем самым использование шаблонов предполагает масштабируемость параллельной программы. В системе Grapnel также проявилась тенденция в развитии визуальных языков параллельного программирования, связанная с тем, что в визуальном виде представляются лишь те части программы, которые имеют отношение к описанию взаимодействия между процессами. Остальные (сравнительно простые для программиста) фрагменты, относящиеся к последовательным частям программы, а также декларативные фрагменты задаются в текстовом виде.Термин “Визуальное программирование” подразумевает использование визуальных выражений (графов, схем, диаграмм, пиктограмм и т.п.) в процессе программирования, что дает возможность специфицировать программу в двумерном или трехмерном виде.

4. Проблемы диаграмматичности

Обзор средств визуального программирования параллельных вычислений показывает, что в большинстве своём словарь их языков основан на диаграммах или схемах различных типов. Чем характеризуются эти языки? Прежде всего, ограниченным набором возможных визуальных элементов языка с жестко определенными значениями и правилами размещения на экране. У языков, использующих потоки управления, в общем-то, нет какого-то смысла, дополнительного по отношению к традиционным текстовым языкам программирования. В диаграмматических системах, основанных на представлении потока управления, обычно отсутствуют средства

графического представления данных программы. То, что касается данных зачастую должно вводиться в обычном текстовом виде. Все эти системы нуждаются в обширном объеме текстового ввода. Использование графов потоков данных влечет аналогичные проблемы - отсутствие высокоуровневых управляющих структур, приводящее к усложнению и запутыванию диаграмм, отсутствие средств поддержки нетривиальных структур данных и т.д. Как уже указывалось, при применении графов потоков данных единственный способ передачи семантической информации - это использование имен узлов и дуг, а у систем, основанных на других схемных метафорах сильно ограничена область применения.

По нашему мнению рано делать какие-либо заключения о пригодности предлагаемых в “концептуальных” средах идей параллельного программирования. Что же касается методик непосредственного создания программ и визуального представления в подобных средах, то все они, несмотря на использование различных подходов к заданию параллелизма, стандартны. Программист создает общую схему параллельной (или распределенной) программы, а затем конкретизирует ее путем задания конкретных деталей на отдельных участках. Образность - диаграмматическая. Набор графических элементов традиционно невелик и, в целом, упрощен, даже, если и используются какие-либо пиктограммы (иконы). Анимация для описания динамики процессов не используется.

Отметим, что ограниченность “диаграмматического” словаря часто не позволяет решать проблемы, проявляющиеся при разработке средств визуального представления в связи с развитием современных языков программирования.

Динамические языки и компилируемые языки последнего поколения развивались, среди прочих направлений, и в направлении развития средств описания процедур доступа к элементам данных. Так, они предоставляют программистам возможность использовать в операторных выражениях обращения к динамическим и ассоциативным массивам, к спискам, к элементам разбиения строк по регулярным выражениям, в некоторых случаях (Ruby) даже к записям в таблицах баз данных.

Обратим внимание на важную для многих диаграмматических языков особенность - исключение из структуры программы механизмов адресации данных, вместо которых предлагается абстракцию *стрелки* (->, <-), связывающей выходы одних операторов (или иных программных конструкций) с входами других. *Стрелка*, однако, не является метафорой доступа к элементу структуры данных задачи, так как не предполагает формирования значения вне связи между двумя операторами, что не соответствует одному из основных свойств данных - существовать и тогда, когда действия над ними не производятся. То есть, в большинстве визуальных языков программист должен иметь дело, хоть и с элементарными, но неявными доступами к значениям (даже не к данным). Развитых средств описания ссылок на результаты вычисления в таких языках нет, и единственной такой ссылкой является узел в орграфе, представляющий оператор, который изображается в виде некоторого трёхмерного или чаще двумерного объекта, например, некоторого многоугольника с исходящей из него линией, символизирующей результат, к которой подключаются входящие в другие узлы-операторы дуги.

Несмотря на то, что при таком подходе существует возможность независимо определять операнды для операторов, которые в этих системах обычно являются n-арными, внесение локальных изменений в программу может потребовать глобального редактирования связей в её графическом представлении. Например, тогда, когда значение, получаемое в одной группе операторов нужно обработать совместно со значением из другой, пространственно отдалённой.

На языке с явным доступом к данным (особенно императивном) в такой ситуации можно было бы обойтись несколькими локальными исправлениями текстового представления программы: сохранить требуемые значения в структуре данных программы, добавив несколько выражений в разнесённые по коду группы операторов, а потом обратится к ним, тогда, когда понадобится их совместная обработка.

Отметим, что иная ситуация возникает у иконических языков, основанных на использовании потоков данных. (Примером подобного языка является визуальный язык HI-Visual, лежащий в основе системы автоматизации конторского труда и основанный на популярной метафоре рабочего стола [10].) В таких языках сначала нужно выбрать группу объектов (операндов), а затем операцию над ними. То есть язык ориентирован на явное указание данных, над которыми нужно выполнить операцию.

Конечно, на популярность и широту применения тех или иных средств программирования влияют многие факторы, но наличие в инструменте программирования развитых средств связи операторной части выражений с данными делает его более удобным в использовании. Эти средства могут играть важную роль в упрощении параллельного программирования.

Во второй половине 90-ых годов был предпринят ряд попыток выхода из “диаграммати-ческого” тупика и ввода в процесс программирования динамики.

Нами был предложен проект визуального языка, использующего некоторый аналог популярных в статистической графике диаграмм Гантта (Gantt charts), традиционно применяемых в системах отладки производительности параллельных программ. Диаграммы Гантта используют полосы различной длины и типа для показа продолжительности того или иного состояния процесса. В нашем случае диаграммы изображались не в виде прямой линии, а в виде цветного двумерного и трехмерного кольца. Был реализован макет двумерного варианта визуального языка параллельного программирования. Пользователь, используя такие понятия, как процесс, обмен, работа, ожидание, канал и др., и задавая времена работы и ожидания, мог определять основные характеристики процессов в специальных окнах. Затем визуально описываются коммуникационные связи между процессами. Уже на ранних этапах описания процесса могло начинаться динамическое отображение смоделированных возможных вариантов его работы в виде модифицированных диаграмм Гантта. Дополнительные описания процесса, как на внутреннем уровне, так и на уровне взаимодействия должны изменять картину, отображающую его деятельность. (Предложенные идеи были близки к идеям, описанным в проекте системы T-Model [11].) Была также сделана попытка создать трехмерный вариант этого языка с использованием размещаемых в пространстве трехмерных аналогов диаграмм Гантта. Несмотря на недостаток серьезной базовой концепции программирования, идеи использования динамики и трехмерности (не привязанной к графово-диаграмматическим канонам) представляются достаточно плодотворными и могут использоваться в дальнейших разработках.

Существуют примеры разработок систем визуализации параллельных вычислений, в которых делается попытка визуализации, как параллелизма, так и динамики обработки данных. (См. среду на базе языка VIM Language [12].) Здесь предусматривается прямое отображение визуальных спецификаций непосредственно в программу конкретного процессора. Визуальные образы должны представлять высокоуровневые математические объекты, например, параметризированную матрицу и вектор при описании методов решения задач линейной алгебры.

Визуальный интерфейс, задающий начальные значения для прикладной вычислительной системы, можно рассматривать как специализированный визуальный язык параллельного программирования. Так, в рамках проекта ASSY [13], который направлен на создание метасистемы, поддерживающей разработку проблемно-ориентированных систем программирования, реализованы средства для решения задачи о взаимодействиях потоков разреженной плазмы. Плазма представляется набором довольно большого числа модельных частиц, которые характеризуются набором параметров, таких как заряд, координаты, скорость и т.п. Область решения разбивается равномерной прямоугольной сеткой на ячейки. С каждой частицей связывается ячейка, в которой частица в данный момент находится. 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.

В подобных средах осуществляется визуализация только лишь основных сущностей программирования, в данном конкретном случае являющимися высокоуровневыми понятиями определенного вычислительного метода. В каком-то смысле можно говорить о реализации визуального интерфейса к вычислительным математическим пакетам.

5. Восприятие визуальных языков

Следует указать также на проблемы, возникающие при визуальном представлении и соответственно адекватном восприятии данных программы и ее динамики.

При проектировании визуальных сред для параллельного программирования (также как и для отладки правильности и производительности программ) необходимо обратить серьезное внимание на проблемы, связанные с восприятием и интерпретацией визуальных образов. Опишем те из них, которые уже как-то прояснились после изучения и анализа существующих систем, а также на основе собственного опыта исследований и экспериментальных разработок.

Визуализация реальных параллельных программ приводит к громоздким и зачастую не интерпретируемым изображениям. Каким бы большим не был экран, объем визуальных данных, необходимых для представления серьезной параллельной или распределенной программы, будет превосходить его возможности. Практика показывает, что даже не очень большое усложнение структуры программы приводит к запутанным картинкам, напоминающим по сложности интерпретации пазлы.

Мы рассматривали различные методы решения этой проблемы. Например, в средах визуального программирования 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 или внутри его [14]. Нами также были реализованы пилотные версии программ с имитацией полета над плоским и сферическим информационным пространством и с возможностью погружения в него. Трехмерные методики размещения объектов параллельных программ использовались нами с середины 90-ых годов. Такие методики предполагают использование тех или иных метафор визуализации параллельных вычислений, в частности, метафоры комнаты. Нами предпринимался ряд попыток реализации метафоры комнаты в макетных системах визуального программирования. В одной из таких систем диаграммы и пиктограммы, размещенные на стенах, представляли управляющие конструкции и данные программной функции. Связи между конструкциями показывалась в пространстве, а не на плоскости, как в традиционных двумерных диаграмматических и иконических системах визуального программирования. Вся программа в этом случае представлялась зданием, а интересующая пользователя комната-функция вытягивалась из здания подобно блоку при блочном строительстве. Однако и в этом случае разработчик, используя в той или иной мере визуализированный инструментарий, должен был создавать программы в традиционном, по сути, текстовом режиме [15].

Методики визуализации на базе виртуальной и расширенной реальности также могут применяться как при создании, так и при отладке параллельных программ. Именно они должны обеспечить наиболее эффективное использование трехмерности и динамики. Однако, кроме непосредственного восприятия, серьезной проблемой остается и адекватная интерпретация чрезвычайно больших объемов сложноструктурированной информации, а также проблема порождения сложных визуальных текстов.

6. Заключение

Следует отметить, что постепенно интерес к созданию визуальных языков параллельного программирования ослабевает. Новые разработки (данные - на начало десятых годов XXI столетия) появляются все реже, хотя говорить о полном прекращении активности в этой области исследований нельзя.

Таким образом, оказывается, что развитие средств визуального программирования параллельных вычислений в своем развитии столкнулось с целым рядом проблем, касающихся как принципиальных вопросов визуального описания сущностей современных языков программирования, так и восприятием больших объемов визуальной информации. Представляется, что решения возникающих проблем следует искать на путях создания принципиально новых методик параллельного программирования, включающих в себя возможность использования метафор разработки и визуализации, способных поддерживать адекватные ментальные модели.

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., с. 68-80.
2. Al-Mulhem M.S. Concurrent programming in VISO // Concurrency: Pract. Exper. 2000; 12, pp. 281-288.
3. Newton P. A Graphical Retargetable Parallel Programming Environment and Its Efficient Implementation // Technical Report TR93-28, Dept. of Computer Sciences, Univ. of Texas at Austin, 1993.
4. Beguelin A.L., Nutt G.J. Visual parallel programming and determinacy: A language specification, an analysis technique, and a programming tool. Journal of Parallel and Distributed Computing, 22(2), August 1994, pp. 235-250.
5. Stankovic N., Zhang K. A distributed parallel programming framework // IEEE Transactions on Software Engineering. Vol. 28. No 5. May 2002, pp. 478-493.
6. Webber J., Lee P.A. Visual, Object-Oriented Development of Parallel Applications // Journal of Visual Languages & Computing Vol. 12, Issue 2, pp. 145-161.
7. Lee P.A. Phillips C., Watson P. Final Report: High Performance (Parallel) Object-Oriented Software Systems (HiPPO) // http://www.parallelism.cs.ncl.ac.uk/projects/hippo/FinalReport.pdf
8. Chan F., Cao J., Chan A.T. S., Zhang K. Visual programming support for graph-oriented parallel/distributed processing // Softw. Pract. Exper. 2005; 35, pp. 1409–1439.
9. Kacsuk P . Dozsa G . Fadgyas T . Designing parallel programs by the graphical language GRAPNEL / Microprocess . And Microprogramm. 1996, V. 41, 8-9, pp. 625 - 643.
10. Hirakawa M., Tanaka M., Ichicawa T. An Iconic Programming System, HI-VISUAL // IEEE Transactions on Software Engineering. Vol. 16 No 10 (October 1990), pp.1178-1184.
11. Самофалов В.В., Коновалов А.В. Т-модель: система наглядных представлений параллельных процессов в транспьютерных сетях // Алгоритмы и программные средства параллельных вычислений. Екатеринбург. ИММ УрО РАН. 1995. Стр. 157-169.
12. Mirenkov N. VIM Language Paradigm // Proceeding of CONPAR-94 - VAPP VI International Conference on Parallel and Vector Processing. Jahannes Kepler University of Linz. Austria. September 6--8 1994. (Lecture Notes in Computer Science) Springer-Verlag. Berlin. 1994. pp. 569-580.
13. Вшивков В.А., Краева М.А., Малышкин В.Э. Параллельная реализация метода частиц // Программирование. 1997, N 2. Стр. 39-51.
14. Флягина Т.А., Авербух В.Л. Новые подходы к проектированию видов отображения и методов интерфейса для систем визуализации программного обеспечения параллельных вычислений // XI Международный семинар Супервычисления и математическое моделирование. Тезисы. Саров. РФЯЦ ВНИИЭФ. 2009. Стр. 106-107.
15. Авербух В.Л., Байдалин А.Ю., Исмагилов Д.Р., Казанцев А.Ю., Тимошпольский С.П., Использование трехмерных метафор визуализации // 14-я Международная Конференция по Компьютерной Графике и Зрению ГрафиКон'2004 6-10 Сентября 2004 Москва, Россия. Труды Конференции. МГУ им. М.В. Ломоносова. Стр. 295-298.