Министерство образования и науки Российской Федерации   
Федеральное государственное автономное государственное учреждение высшего

профессионального образования  
«**Уральский федеральный университет**

**имени первого Президента России Б. Н. Ельцина**»   
  
Математико-механический факультет   
Кафедра информатики и процессов управления

Разработка методик визуализации для представления

работы параллельных программ

«Допускается к защите» Квалификационная работа на степень бакалавра наук  
 Заведующий кафедрой студента группы МТ - 405  
 УРОСОВА АЛЕКСАНДРА ПАВЛОВИЧА

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2011 г.

Научный руководитель

АВЕРБУХ ВЛАДИМИР ЛАЗАРЕВИЧ,

заведующий сектором компьютерной визуализации

ИММ УрО РАН, доцент КИПУ, кандидат технических наук

Екатеринбург

2011

Реферат

Уросов А. П. РАЗРАБОТКА МЕТОДИК ВИЗУАЛИЗАЦИИ ДЛЯ ПРЕДСТАВЛЕНИЯ РАБОТЫ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ, квалификационная работа на степень бакалавра наук.

Ключевые слова: Визуализация параллельных вычислений, Система RiDE, Параллельное программирование.

Объект исследования: система RiDE, разрабатываемая для программирования в параллельных распределённых средах.

Цель работы: разработка методик визуализации для представления работы параллельных программ, написанных для системы RiDE. Разработка программы-визуализатора.

В процессе работы проводился анализ возможностей для визуализации работы программы, написанной для системы RiDE. Выбирались сущности и события, на основе которых можно сделать комплексную визуализацию работы программы.

В результате работы разработана программа, предназначенная для визуализации работы параллельной программы для системы RiDE.

Содержание

ВВЕДЕНИЕ………………………………………………………………………………………………………..4

ВИЗУАЛИЗАЦИЯ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ…………………………………………..6

СИСТЕМА RiDE………………………………………………………………………………………………….

МЕТОДИКА ВИЗУАЛИЗАЦИИ…………………………………………………………………………..

ОПИСАНИЕ ВИЗУАЛИЗАТОРА………………………………………………………………………….

ЗАКЛЮЧЕНИЕ…………………………………………………………………………………………………..

ЛИТЕРАТУРА…………………………………………………………………………………………………….

Введение

Современный уровень развития информационных технологий требует принципиально новых средств разработки программного обеспечения. Переломным моментом стало появление на массовом рынке персональных компьютеров, имеющих многоядерные центральный и графический процессоры. Теперь, чтобы написать программу, эффективно использующую ресурсы компьютера, программисту необходимо также заботиться и о ее распараллеливании. Однако написание параллельной программы является значительно более сложным процессом, чем традиционное последовательное программирование. Усложняется и отладка программы, так как возникают новые классы ошибок, связанные, во-первых, с множеством отдельных потоков управления (процессов), которые выполняются одновременно, а во-вторых, с асинхронными взаимодействиями этих потоков управления друг с другом. В программировании для суперкомпьютеров и рабочих станций, которые изначально строились как параллельные многопроцессорные системы, так же произошли качественные изменения. Связаны они как с существенным увеличением количества процессоров, так и с применением процессоров разного типа. Современный суперкомпьютер может насчитывать десятки тысяч как многоядерных центральных процессоров (CPU), так и графических процессоров (GPU), которые в свою очередь так же являются сложными параллельными процессорами, состоящими из тысяч вычислительных блоков. Разработка программного обеспечения для таких гибридных систем сильно осложняется большим количеством вычислительных блоков, а так же, что более существенно, различиями в подходах к программированию для центральных и графических процессоров. Вследствие этого очень остро встают проблемы сложности разработки и контроля правильности выполнения программы. Решение этих проблем требует комплексного подхода, включающего в себя создание средств разработки, отладки и контроля выполнения программы. Поэтому создание средств визуализации для подобных систем является очень актуальной задачей.

Моя работа относится к области визуализации программного обеспечения параллельных и распределенных систем. Использование средств визуализации повышает эффективность разработки (средства визуального программирования) и отладки (визуальные отладчики) параллельных программ. Существующие программные среды параллельных вычислений основываются по большей части на парадигме передачи сообщений или использования общей памяти. В то же время идут разработки других систем, основывающихся на модели потока данных. В частности, исследования по таким программным средам проводятся в Институте математики и механики Уральского отделения РАН. Моей задачей была разработка методик визуализации для разрабатываемой в ИММ УрО РАН системы RiDE, предназначенной для программирования в параллельных распределённых средах.

Визуализация параллельных вычислений

Система RiDE

Распространенные средства параллельного программирования, такие как MPI и OpenMP, требуют от программиста подробного описания большого количества сущностей. Необходимо заботиться о распределении вычислительных задач, синхронизации и обмене данными. В связи с этим, программирование с использованием этих средств является трудоемкой задачей, занимающей существенную часть рабочего времени. Программы создаются долго, получаемые коды сложны и громоздки, их сопровождение и развитие оказывается затратным процессом.

Другой проблемой распространенных средств параллельного программирования является их ориентация на конкретные классы вычислительных систем, таких как системы с общей памятью, кластерные системы или распределенные системы. Программы, написанные с помощью таких средств, способны выполняться на предназначенном для них типе параллельной системы, но не способны эффективно работать с системой другого типа. Кроме того, данные средства, как правило, не содержат встроенной поддержки ускорителей (GPU, ПЛИС и т.д.), а также концепций облачных вычислений и SaaS.

Поэтому особый интерес представляет создание такого универсального средства, которое бы обеспечивало возможность разработки и эффективного исполнения программ во всех типах вычислительных сред, а также в их смешанных конфигурациях.

RIDE - это методика и основанная на ней система для программирования в параллельных распределённых средах. Главные критерии, на которые ориентировались авторы при ее разработке, следующие:

1. Методика должна предоставить универсальный механизм параллельного программирования, болеепростой в применении, чем существующие универсальные средства.
2. Вычислительные программы, реализованные с помощью разрабатываемой методики, должны выполняться не менее эффективно, чем при использовании других средств.
3. Методика должна ориентироваться на создание и эффективное исполнение программ на всех типах вычислительных систем, в первую очередь в распределенных вычислительных средах.

Методика базируется на понятиях ***хранилища***, ***задач*** и ***правил***.

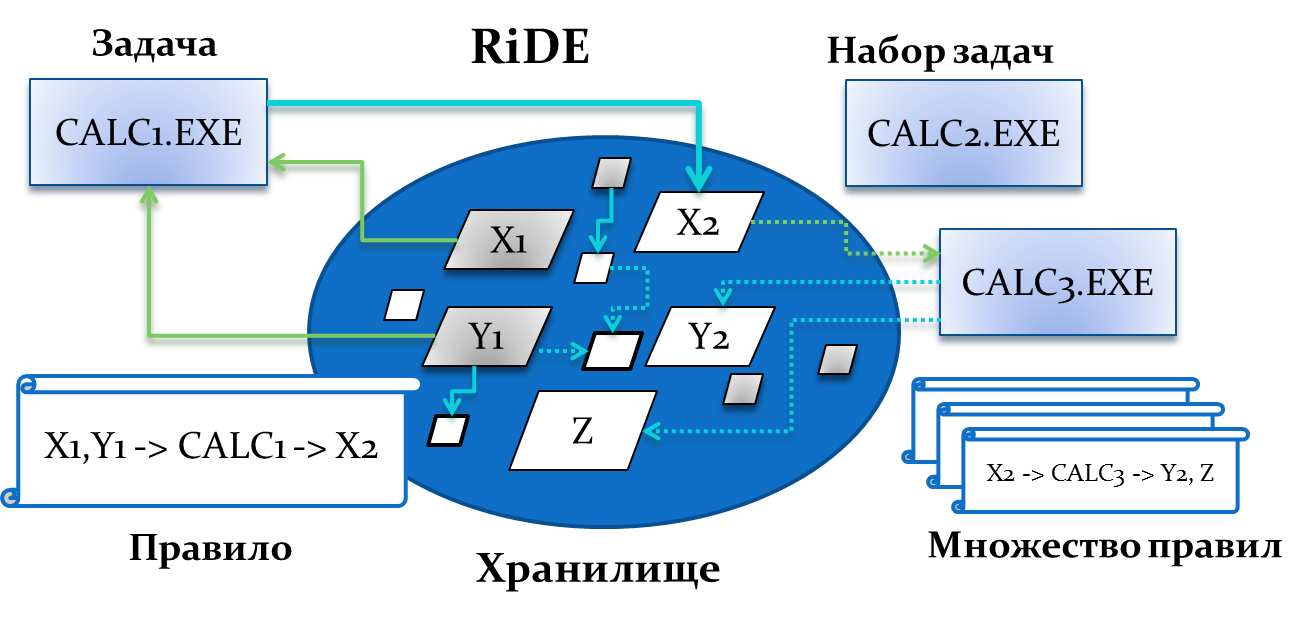
***Хранилище*** содержит в себе именованные данные, по отношению к которым доступны три операции – создание (запись), чтение и удаление. Хранимые данные являются самодостаточными - это не очереди, но некие цельные единицы информации с уникальными именами. Допускаются операции частичного чтения данных.

***Задачей*** называется программа, которая во время исполнения считывает данные с определенными именами из *хранилища* и в результате своего исполнения формирует новые данные, которые записываются в хранилище.

***Правилом*** называется такая конструкция, которая определяет условия и параметры запуска *задач*. Правило содержит в себе:

1. Список имен данных, которые необходимы для выполнения задачи.
2. Список соответствия глобальных имен данных (находящихся в хранилище) локальным именам (с которыми и будет работать задача).
3. Список задач (программ), которые необходимо запустить.
4. Действия, совершаемые в случае успешного выполнения задач (3).

Правило считается готовым к исполнению, когда в хранилище присутствуют все данные с именами из списка (1). После успешного исполнения правило удаляется из списка выполняемых правил.



**Рис. 1**. Участники процесса вычислений в RiDE

На **Рис. 1** представлена общая схема участников описываемой методики. В центре находится хранилище. Белым выделены данные, которых в хранилище пока нет, серым - которые уже есть. Показан пример правила, которое гласит: при наличии данных X1 и Y1 необходимо запустить программу Calc1.exe, подать ей на вход эти данные, а результат работы записать в данные с именем X2. Показано и другое правило, которое требует выполнить другую программу при наличии элемента данных с именем X2. Очевидно, что это правило сработает только после того, как будет завершено первое правило. Это отмечено с помощью пунктира, который говорит, что обозначенный запуск и как результат порождение новых данных пока невозможно, но свершится в будущем. Вполне вероятно, что на рисунке есть и правила, которые могут выполняться независимо от представленных двух. Степень независимости и определяет меру параллелизма, с которой может быть произведено вычисление.

Процесс программирования и проведения вычислений в рамках RiDE происходит следующим образом. Прежде всего, разрабатываются программные коды задач, из которых состоит вычислительный эксперимент. Каждая такая задача на этапе инициализации должна считать данные из хранилища, а затем по ходу выполнения сформировать и записать новые данные в хранилище. Отметим, что в рамках одного вычисления могут использоваться любые комбинации языков, а также целевых аппаратных сред для создания задач. Например, часть задач можно реализовать на графических ускорителях, а часть – на обычных процессорах.

После создания вычислительных программ (задач) программистом формируется файл инициализации, в котором описываются начальные правила системы. В дальнейшем эти правила могут дополняться – при выполнении задач или финализации правил. Кроме правил, в файле инициализации указываются начальные данные, которые помещаются в хранилище.

После подачи команды на запуск вычислительная среда ищет правила, готовые к исполнению, и запускает указанные в них задачи на подходящих свободных вычислительных ресурсах. В результате часть правил исполняется, формируя новые данные и освобождая ресурсы для других правил. Среда продолжает поиск и выполнение правил вплоть до исчерпания всех правил, приостановки работы с внешней стороны или выявления ошибки.

Программная реализация предлагаемой методики способна обеспечить следующие преимущества:

* Разделение уровней вычисления и взаимодействия, что обеспечивает более ясный процесс создания, отладки и сопровождения вычислительных программ. Вычислительные коды группируются в задачах; коммуникационные – в описаниях правил.
* Поддержка всех типов вычислительных сред – с общей памятью, кластерных и распределенных. Работа во всех средах может быть реализована эффективно, без необходимости переработки вычислительных программ, включая системы с общей памятью, где чтение и запись данных могут быть реализованы без накладных операций копирования.
* Возможность включения и отключения вычислительных ресурсов «на лету», что позволяет максимально эффективно задействовать вычислительные ресурсы и гибко планировать их распределение. Это свойство обеспечивается тем, что все обмены данными происходят через интерфейсы хранилища.
* Использование любых языков программирования для создания вычислительных программ. В программах должны присутствовать только два типа RiDE-функций – чтение и запись данных в хранилище.
* Отсутствие ограничений на внутреннюю сложность задач, вызываемых при срабатывании правил – задача, например, сама может быть параллельной MPI-программой или даже закрытой коммерческой программой, написанной вне рамок предлагаемой методики. В последнем случае взаимодействие программы с RiDE-окружением реализуется через файлы.
* Возможность участия в рамках одного вычисления программ различных платформ (различные ОС, языки программирования, процессоры и ускорители). Назначать правила на исполнение можно сообразно требованиям кодов задач к аппаратным характеристикам вычислительных узлов. Более того, в правиле можно указать различные версии вычислительных программ, написанные для разных целевых платформ; выбор конкретной версии может осуществляться исходя из имеющихся свободных вычислительных ресурсов.
* Возможность реализации поддержки программирования ускорителей. Программные средства уровня OpenCL очень сложны в использовании; в рамках методики можно реализовать механизмы, упрощающие загрузку-выгрузку данных из вычислительных устройств;
* Встроенная поддержка контрольных точек. Весь обмен данными, и таким образом, текущее состояние счета, размещается в хранилище. Снимок состояния хранилища и формирует контрольную точку. При этом состоянием оперативной памяти считающихся в текущий момент задач можно пренебречь, осуществив при необходимости их пересчет.
* Возможность полностью остановить счет и в будущем продолжить его. Продолжение счета может быть осуществлено на других вычислительных ресурсах.
* Автоматическая оптимизация вычислений путем оптимального размещения правил на вычислительных узлах. Распределение правил по узлам может осуществляться на основе анализа статистики по предыдущим запускам, текущего размещения данных, зависимостей данных, описанных в правилах (заметим, что это описание – явное).

Методика визуализации

Исходя из подробного анализа описания системы RiDE, можно разработать методику, основанную на визуализации базовых для нее понятий хранилища, задач и правил.

В первую очередь необходимо реализовать отображение хранилища. Хранилище является одним из главных базовых понятий системы и должно занимать в визуализации центральное место. Само понятие хранилища является абстрактным, то есть нас не интересует его конкретная реализация, поэтому мы будем отображать его как неупорядоченный набор данных. Хранилище размещается в центре экрана и при полном заполнении данными имеет форму квадратной матрицы. При неполном заполнении в хранилище появляются пустые места, которые могут быть в любом месте. Связано это с возможностью удаления более не нужных данных из хранилища. Система RiDE не выполняет эту процедуру автоматически, поэтому о “сборке мусора” необходимо заботиться программисту самостоятельно. При добавлении новых данных в хранилище они размещаются по порядку на свободные места, начиная с левого верхнего угла. Безусловно, такая форма представления хранилища не имеет ничего общего с его реальным устройством, однако обеспечивает

С понятием хранилища неразрывно связано понятие данных, которые в нем содержатся. Было решено отображать данные как маленькие цветные шарики с границей. Такое представление данных очень наглядно:

Описание визуализатора

Программа-визуализатор RideVis написана на языке C# с использованием технологии Windows Presentation Foundation (WPF). Для ее работы необходима операционная система Microsoft Windows и программная платформа Microsoft .NET Framework 3.5 или выше. Окно программы представлено на рисунке 1.



Рисунок 1

Интерфейс программы состоит из трех полей для ввода параметров и кнопки начала отображения визуализации. Первым параметром задается путь к файлу с историей выполнения программы для системы RiDE. По умолчанию размещается в текущем каталоге с программой. Данный файл создается самой системой RiDE при выполнении некоторой программы, процесс работы которой необходимо визуализировать. Вторым параметром задается интервал времени в миллисекундах, в течение которого прочитанные данные находятся вокруг процессов, после чего исчезают. В качестве третьего параметра выступает скорость визуализации. Это число с плавающей запятой: 0,1 0,5 1 2 и т. д. Устанавливает, во сколько раз скорость визуализации будет отличаться от номинальной (от 1).

Рассмотрим формат файла с историей выполнения программы. В первой строке задается количество процессоров в системе. Далее следует описание начального состояния хранилища: во второй строке указывается количество начальных данных в хранилище и в следующих строках перечислены их имена по одному имени в каждой строке. Все имена должны быть уникальными. Пример:

4 - количество ядер в системе

36 - количество данных в хранилище

Test1 - имена данных в хранилище

Test2

Test3

Test4

Test5

Test6

Test7

………

Test36

Далее перечислены события, произошедшие в ходе выполнения визуализируемой программы. Эти события являются командами для программы-визуализатора. Общий формат команды имеет следующий вид:

время в миллисекундах от старта программы | тип события | дополнительные параметры | дополнительные параметры | …

Для каждого типа событий предусмотрен свой формат команды. Подробно рассмотрим каждый из них:

1. “ps” запустился процесс с некоторым именем на некотором процессоре (процессоры нумеруются с 0):

время|тип события|имя процесса|номер процессора

Пример:

1000|ps|Prosess1|0

2000|ps|Prosess2|2

3000|ps|Prosess3|1

1. “pt” процесс завершился:

время|тип события|имя процесса

Пример:

11000|pt|Prosess3

12000|pt|Prosess1

13000|pt|Prosess4

1. “pr” процесс прочитал данные из хранилища:

время|тип события|имя процесса|имя данного из хранилища

Пример:

4340|pr|Prosess3|Data1

4350|pr|Prosess3|Data2

4360|pr|Prosess3|Data3

1. “pw” процесс записал данные в хранилище:

время|тип события|имя процесса|имя данного

Пример:

5340|pw|Prosess2|Data42

5350|pw|Prosess2|Data43

5360|pw|Prosess1|Data44

1. “da” данные поступили в хранилище откуда-то извне:

время|тип события|имя данного

Пример:

8000|da|Data37

9000|da|Data38

1. “dd” данные удалены из хранилища:

время|тип события|имя данного из хранилища

Пример:

10000|dd|Data1

В каждый момент времени имена данных в хранилище должны быть уникальными.

Перейдем к рассмотрению визуализации работы программы. Рассмотрим Рисунок 2. В центре окна размещается хранилище, состоящее из набора начальных данных. Начальные данные отображаются как маленькие белые шарики с черной границей. Вокруг хранилища показаны орбиты, количество которых совпадает с количеством процессоров в системе. Каждая орбита соответствует одному определенному процессору и не меняет свое значение в процессе работы.

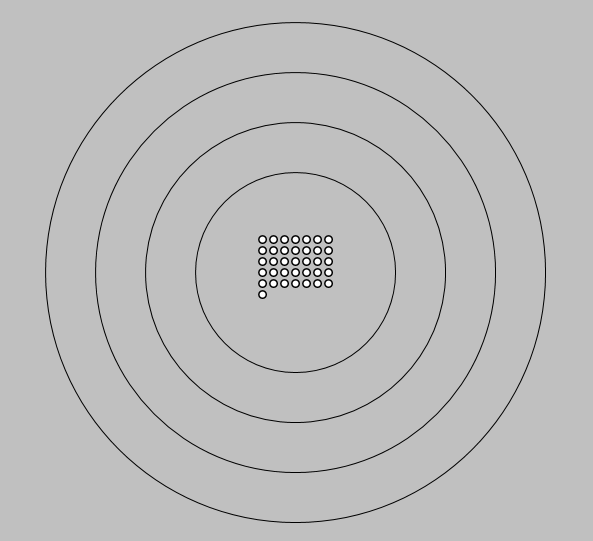


Рисунок 2

Для того чтобы программа выполнялась должен быть запущен минимум один процесс. Рассмотрим рисунок 3. Процессы отображаются как большие цветные шарики с черной границей. Цвет каждого процесса является уникальным и остается таким даже после его завершения. Запуск процесса с некоторым именем на определенном процессоре отображается как появление шарика определенного цвета на нужной орбите. Выполнение процесса показано движением шарика по орбите и взаимодействием с данными в хранилище.

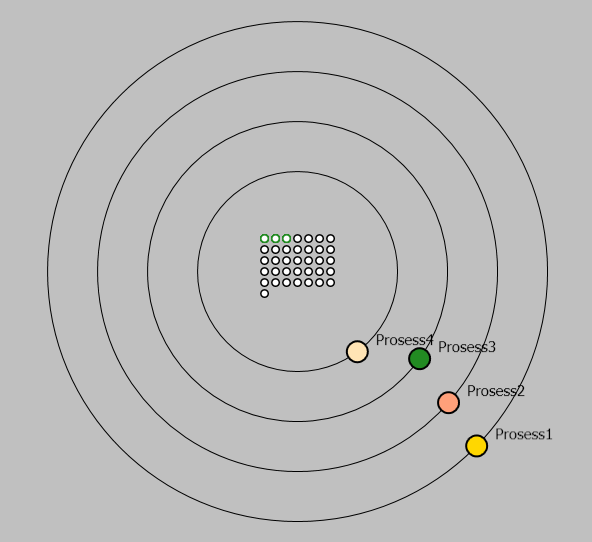


Рисунок 3

Чтение данных из хранилища визуализируется следующим образом: процесс подсвечивает границы читаемых данных своим цветом, и их копии вылетают из хранилища и прикрепляются к процессу, как показано на Рисунке 4. После некоторого времени, задаваемого вторым параметром программы, эти данные исчезают. Глядя на количество прикрепленных к процессу данных за указанный промежуток времени, можно определить какие процессы наиболее интенсивно читают данные из хранилища.

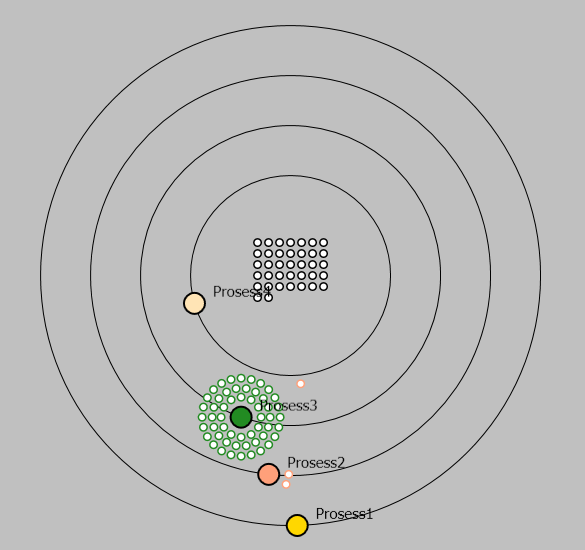


Рисунок 4

Запись данных в хранилище проиллюстрирована на Рисунке 5. Из центра процесса вылетает маленький шарик с черной границей и того же цвета, что и создавший его процесс. Записанные данные размещаются на свободных местах в хранилище и сохраняют свой цвет до завершения работы программы. Так как каждый процесс имеет свой уникальный цвет, то глядя на состояние хранилища можно определить какие процессы наиболее интенсивно записывают в хранилище данные. Когда процесс завершается, то соответствующий ему шарик исчезает.

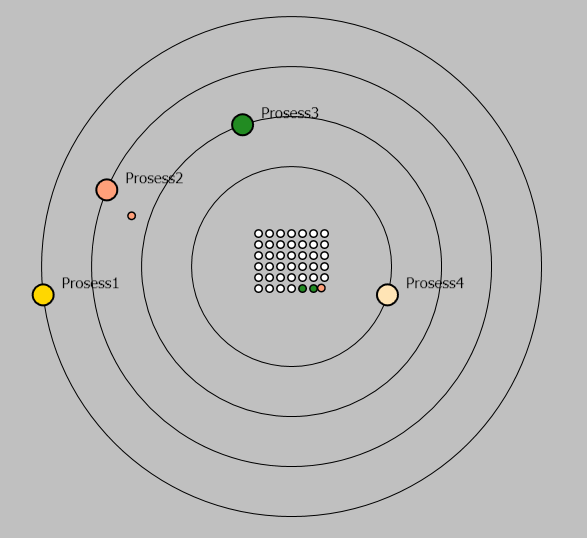


Рисунок 5

Дополнительные операции работы с хранилищем, такие как добавление новых данных в хранилище и удаление данных из него, визуализируются достаточно просто: шарик либо появляется на свободном месте в хранилище, либо исчезает из хранилища, освобождая место.

Кратко рассмотрим структуру самой программы. Она состоит из двух частей: интерфейса и кода отображения визуализации. Интерфейс написан на языке XAML - основанном на XML языке разметки для декларативного программирования приложений. XAML разработан компанией Microsoft. Исходный код приведен далее:

<Window x:Class="Diplom.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="MainWindow" Height="701" Width="701" Background="Silver" >

<Grid Name="grid1">

<Label Content="Путь к файлу данных:" Height="28" HorizontalAlignment="Left" Margin="12,12,0,0" Name="label1" VerticalAlignment="Top" />

<TextBox Height="23" HorizontalAlignment="Left" Margin="141,12,0,0" Name="textBox1" VerticalAlignment="Top" Width="125" Text="in.txt" />

<Label Content="Интервал:" Height="28" HorizontalAlignment="Left" Margin="272,12,0,0" Name="label2" VerticalAlignment="Top" />

<TextBox Height="23" HorizontalAlignment="Left" Margin="340,12,0,0" Name="textBox2" VerticalAlignment="Top" Width="60" Text="5000" />

<Label Content="Скорость:" Height="28" HorizontalAlignment="Left" Margin="406,10,0,0" Name="label3" VerticalAlignment="Top" />

<TextBox Height="23" Margin="472,12,0,0" Name="textBox3" VerticalAlignment="Top" Text="1" HorizontalAlignment="Left" Width="60" />

<Button Content="Старт" Height="23" HorizontalAlignment="Left" Margin="549,12,0,0" Name="button1" VerticalAlignment="Top" Width="118" Click="Visualisation\_start" />

<Canvas Name="Base">

</Canvas>

</Grid>

</Window>

Здесь создается главное окно программы Window с различными параметрами, далее создается контейнер Grid для размещения в нем интерфейса программы, и в самом конце размещается контейнер Canvas для отображения в нем визуализации.

Код отображения визуализации написан на языке C#. Далее приведен исходный код:

using System;

using System.IO;

using System.Collections.Generic;

using System.Text;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using System.Windows.Media.Animation;

using System.Windows.Threading;

using System.Threading;

namespace Diplom

{

class Processes

{

public string Name;

public Label Text;

public Ellipse myEllipse;

public OrbitClass Orbits;

public Processes()

{

Orbits = new OrbitClass(10);

}

}

class Data

{

public string Name;

public Ellipse myEllipse;

}

class FlyingData

{

public Data data;

public int pause;

public string destination;

public int vaultPosition;

public int coreNumber;

public orbitPosition op;

public FlyingData()

{

data = new Data();

}

}

class orbitPosition

{

public int orbit;

public int position;

public orbitPosition(int i, int j)

{

orbit = i;

position = j;

}

}

class OrbitClass

{

public double center\_x = 0;

public double center\_y = 0;

public Data[][] orbits;

public bool[][] availableOrbits;

public long[][] addTimes;

public double[] alfas;

public double[] radiuses;

public OrbitClass(int numOrbits)

{

orbits = new Data[numOrbits][];

availableOrbits = new bool[numOrbits][];

addTimes = new long[numOrbits][];

alfas = new double[numOrbits];

radiuses = new double[numOrbits];

int size = 12;

for (int j = 0; j < orbits.Length; j++)

{

Data[] orbit = new Data[size];

bool[] availableOrbit = new bool[size];

long[] addTime = new long[size];

for (int i = 0; i < orbit.Length; i++)

{

orbit[i] = null;

availableOrbit[i] = true;

addTime[i] = 0;

}

orbits[j] = orbit;

availableOrbits[j] = availableOrbit;

addTimes[j] = addTime;

alfas[j] = 2 \* Math.PI / size;

radiuses[j] = Math.Ceiling((MainWindow.dataSizeX + 1) / Math.Sqrt(2 \* (1 - Math.Cos(alfas[j]))));

size = size + 6;

}

}

public orbitPosition Add()

{

for (int i = 0; i < availableOrbits.Length; i++)

{

for (int j = 0; j < availableOrbits[i].Length; j++)

{

if (availableOrbits[i][j] == true)

{

availableOrbits[i][j] = false;

return new orbitPosition(i, j);

}

}

}

return null;

}

public void setCenter(double center\_x, double center\_y)

{

this.center\_x = center\_x;

this.center\_y = center\_y;

}

public double getTextX()

{

return center\_x + 25;

}

public double getTextY()

{

return center\_y - 15;

}

public double getX(orbitPosition op)

{

return center\_x + 7 + radiuses[op.orbit] \* Math.Cos(alfas[op.orbit] \* op.position);

}

public double getY(orbitPosition op)

{

return center\_y + 7 + radiuses[op.orbit] \* Math.Sin(alfas[op.orbit] \* op.position);

}

}

class VaultClass // Класс для хранилища

{

int numRows;

int nullX;

int nullY;

int offsetX = MainWindow.dataSizeX + 2;

int offsetY = MainWindow.dataSizeY + 2;

public Data[] vault;

public bool[] availableVault;

public VaultClass(int size)

{

vault = new Data[size];

for (int i = 0; i < vault.Length; i++)

{

vault[i] = null;

}

availableVault = new bool[size];

for (int i = 0; i < availableVault.Length; i++)

{

availableVault[i] = true;

}

numRows = (int)Math.Ceiling(Math.Sqrt(size));

nullX = -(offsetX \* numRows - 2) / 2;

nullY = (offsetY \* numRows - 2) / 2;

}

public int? Add(Data data)

{

for (int i = 0; i < availableVault.Length; i++)

{

if (availableVault[i] == true)

{

availableVault[i] = false;

vault[i] = data;

return i;

}

}

return null;

}

public int? Add()

{

for (int i = 0; i < availableVault.Length; i++)

{

if (availableVault[i] == true)

{

availableVault[i] = false;

return i;

}

}

return null;

}

public Data get(string name)

{

for (int i = 0; i < vault.Length; i++)

{

if (vault[i].Name == name)

{

return vault[i];

}

}

return null;

}

public void delete(string name)

{

for (int i = 0; i < vault.Length; i++)

{

if (vault[i] != null && vault[i].Name == name)

{

availableVault[i] = true;

vault[i] = null;

}

}

}

public int getX(int i)

{

return nullX + (i % numRows) \* offsetX;

}

public int getY(int i)

{

return nullY - (i / numRows) \* offsetY;

}

}

/// <summary>

/// Логика взаимодействия для MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

static int center\_x = 350; // Координаты центра

static int center\_y = 350; // Координаты центра

public static int dataSizeX = 9;

public static int dataSizeY = 9;

public static int processesSizeX = 23;

public static int processesSizeY = 23;

static int numCores; // Количество ядер в системе

static int interval; // Интервал времени для оценки работы процессов

static double visualisationSpeed;

static VaultClass Vault; // Хранилище

Processes[] Cores; // Ядра

static double V = 0.5;

static double[] alfa;

static double[] alfaV;

static DispatcherTimer dispatcherTimer;

static List<FlyingData> flyingData;

static Queue<string[]> commands;

static Stack<Color> prosessColors;

static long counter;

static int speed = 10;

static double getX(double x)

{

return center\_x + x;

}

static double getY(double y)

{

return center\_y - y;

}

public MainWindow()

{

InitializeComponent();

}

public void Visualisation\_start(object sender, RoutedEventArgs e)

{

Base.Children.Clear();

counter = 0;

if (dispatcherTimer != null) dispatcherTimer.Stop();

string tb1 = textBox1.Text.Trim();

if (File.Exists(tb1))

{

prosessColors = new Stack<Color>();

prosessColors.Push(Colors.Brown);

prosessColors.Push(Colors.Green);

prosessColors.Push(Colors.Purple);

prosessColors.Push(Colors.Red);

prosessColors.Push(Colors.Yellow);

prosessColors.Push(Colors.Thistle);

prosessColors.Push(Colors.Teal);

prosessColors.Push(Colors.SandyBrown);

prosessColors.Push(Colors.Violet);

prosessColors.Push(Colors.PowderBlue);

prosessColors.Push(Colors.Olive);

prosessColors.Push(Colors.Moccasin);

prosessColors.Push(Colors.ForestGreen);

prosessColors.Push(Colors.LightSalmon);

prosessColors.Push(Colors.Gold);

string tb2 = textBox2.Text.Trim();

if (tb2 != "")

{

interval = Convert.ToInt32(tb2);

}

string tb3 = textBox3.Text.Trim();

if (tb3 != "")

{

visualisationSpeed = Convert.ToDouble(tb3);

}

StreamReader fin = new StreamReader(tb1);

numCores = Convert.ToInt32(fin.ReadLine());

alfa = new double[numCores];

for (int i = 0; i < alfa.Length; i++)

{

alfa[i] = 0;

}

alfaV = new double[numCores];

for (int i = 0; i < alfaV.Length; i++)

{

alfaV[i] = V / (100 + 50 \* i);

}

Cores = new Processes[numCores];

for (int i = 0; i < alfaV.Length; i++)

{

Cores[i] = null;

}

flyingData = new List<FlyingData>();

commands = new Queue<string[]>();

Queue<string> baseVaultData = new Queue<string>();

//int vaultSize;

int N = Convert.ToInt32(fin.ReadLine());

for (int i = 0; i < N; i++)

{

baseVaultData.Enqueue(fin.ReadLine());

}

int vaultSize = N;

int vaultMaxSize = N;

while (!fin.EndOfStream)

{

string[] readData = fin.ReadLine().Trim().Split('|');

commands.Enqueue(readData);

switch (readData[1])

{

case "da":

vaultMaxSize++;

if (vaultMaxSize > vaultSize) vaultSize = vaultMaxSize;

break;

case "pw":

vaultMaxSize++;

if (vaultMaxSize > vaultSize) vaultSize = vaultMaxSize;

break;

case "dd":

vaultMaxSize--;

break;

}

}

fin.Close();

Vault = new VaultClass(vaultSize); // Создаем хранилище

for (int i = 0; i < numCores; i++)

{

Ellipse myEllipse = new Ellipse();

myEllipse.StrokeThickness = 1;

myEllipse.Stroke = Brushes.Black;

myEllipse.Width = 201 + 100 \* i;

myEllipse.Height = 201 + 100 \* i;

Base.Children.Add(myEllipse);

Canvas.SetLeft(myEllipse, getX(-100 - 50 \* i));

Canvas.SetTop(myEllipse, getY(100 + 50 \* i));

}

for (int i = 0; i < N; i++)

{

string dataName = baseVaultData.Dequeue(); // Считываем имена данных

Ellipse myEllipse = new Ellipse();

SolidColorBrush mySolidColorBrush = new SolidColorBrush();

mySolidColorBrush.Color = Colors.White;

myEllipse.Fill = mySolidColorBrush;

myEllipse.StrokeThickness = 1.5;

myEllipse.Stroke = Brushes.Black;

myEllipse.Width = 1;

myEllipse.Height = 1;

Data myEllipseData = new Data();

myEllipseData.Name = dataName;

myEllipseData.myEllipse = myEllipse;

int position = (int)Vault.Add(myEllipseData);

Base.Children.Add(Vault.vault[position].myEllipse);

Canvas.SetLeft(Vault.vault[position].myEllipse, getX(Vault.getX(position)));

Canvas.SetTop(Vault.vault[position].myEllipse, getY(Vault.getY(position)));

DoubleAnimation widthAnimation = new DoubleAnimation();

widthAnimation.To = dataSizeX;

widthAnimation.Duration = TimeSpan.FromSeconds(1);

MainWindow.Vault.vault[position].myEllipse.BeginAnimation(Ellipse.WidthProperty, widthAnimation);

DoubleAnimation heightAnimation = new DoubleAnimation();

heightAnimation.To = dataSizeY;

heightAnimation.Duration = TimeSpan.FromSeconds(1);

MainWindow.Vault.vault[position].myEllipse.BeginAnimation(Ellipse.HeightProperty, heightAnimation);

}

dispatcherTimer = new DispatcherTimer();

dispatcherTimer.Tick += new EventHandler(dispatcherTimer\_Tick);

dispatcherTimer.Interval = new TimeSpan(0, 0, 0, 0, speed);

dispatcherTimer.Start();

}

}

public void dispatcherTimer\_Tick(object sender, EventArgs e)

{

counter += speed;

// Раздача заданий

while (commands.Count != 0 && (Convert.ToInt64(commands.Peek()[0]) / visualisationSpeed) < counter)

{

string[] currentCommand = commands.Dequeue();

string commandType = currentCommand[1];

Ellipse myEllipse;

SolidColorBrush mySolidColorBrush;

string prosessName;

string dataName;

int coreNumber;

DoubleAnimation widthAnimation;

DoubleAnimation heightAnimation;

switch (commandType)

{

case "ps":

prosessName = currentCommand[2];

coreNumber = Convert.ToInt32(currentCommand[3]);

Processes Prosess = new Processes();

myEllipse = new Ellipse();

mySolidColorBrush = new SolidColorBrush();

mySolidColorBrush.Color = prosessColors.Pop();

myEllipse.Fill = mySolidColorBrush;

myEllipse.StrokeThickness = 2;

myEllipse.Stroke = Brushes.Black;

myEllipse.Width = 1;

myEllipse.Height = 1;

Base.Children.Add(myEllipse);

Canvas.SetLeft(myEllipse, center\_x - 11 + (100 + 50 \* coreNumber));

Canvas.SetTop(myEllipse, center\_y - 11);

widthAnimation = new DoubleAnimation();

widthAnimation.To = processesSizeX;

widthAnimation.Duration = TimeSpan.FromSeconds(1);

myEllipse.BeginAnimation(Ellipse.WidthProperty, widthAnimation);

heightAnimation = new DoubleAnimation();

heightAnimation.To = processesSizeY;

heightAnimation.Duration = TimeSpan.FromSeconds(1);

myEllipse.BeginAnimation(Ellipse.HeightProperty, heightAnimation);

Prosess.Name = prosessName;

Prosess.myEllipse = myEllipse;

Prosess.Orbits.setCenter(Canvas.GetLeft(myEllipse), Canvas.GetTop(myEllipse));

Prosess.Text = new Label();

Prosess.Text.Content = Prosess.Name;

Prosess.Text.FontSize = 15;

Base.Children.Add(Prosess.Text);

Canvas.SetLeft(Prosess.Text, Prosess.Orbits.getTextX());

Canvas.SetTop(Prosess.Text, Prosess.Orbits.getTextY());

Canvas.SetZIndex(Prosess.Text, 100);

Cores[coreNumber] = Prosess;

break;

case "pt":

prosessName = currentCommand[2];

for (int i = 0; i < Cores.Length; i++)

{

if (Cores[i] != null && Cores[i].Name == prosessName)

{

for (int j = 0; j < Cores[i].Orbits.orbits.Length; j++)

{

for (int k = 0; k < Cores[i].Orbits.orbits[j].Length; k++)

{

if (Cores[i].Orbits.orbits[j][k] != null)

{

Base.Children.Remove(Cores[i].Orbits.orbits[j][k].myEllipse);

}

}

}

Base.Children.Remove(Cores[i].Text);

Base.Children.Remove(Cores[i].myEllipse);

Cores[i] = null;

}

}

break;

case "pr":

prosessName = currentCommand[2];

dataName = currentCommand[3];

for (int i = 0; i < Cores.Length; i++)

{

if (Cores[i] != null && Cores[i].Name == prosessName)

{

FlyingData dataFly = new FlyingData();

dataFly.coreNumber = i;

dataFly.op = Cores[i].Orbits.Add();

dataFly.pause = 0;

dataFly.destination = "orbit";

Data getData = Vault.get(dataName);

dataFly.data = new Data();

dataFly.data.Name = getData.Name;

myEllipse = new Ellipse();

mySolidColorBrush = new SolidColorBrush();

mySolidColorBrush.Color = Colors.White;

myEllipse.Fill = mySolidColorBrush;

myEllipse.StrokeThickness = 1.5;

myEllipse.Stroke = Cores[i].myEllipse.Fill;

myEllipse.Width = dataSizeX;

myEllipse.Height = dataSizeY;

Base.Children.Add(myEllipse);

Canvas.SetLeft(myEllipse, Canvas.GetLeft(getData.myEllipse));

Canvas.SetTop(myEllipse, Canvas.GetTop(getData.myEllipse));

dataFly.data.myEllipse = myEllipse;

flyingData.Add(dataFly);

break;

}

}

break;

case "pw":

prosessName = currentCommand[2];

dataName = currentCommand[3];

for (int i = 0; i < Cores.Length; i++)

{

if (Cores[i] != null && Cores[i].Name == prosessName)

{

FlyingData dataFly = new FlyingData();

dataFly.vaultPosition = (int)Vault.Add();

dataFly.pause = 0;

dataFly.destination = "vault";

dataFly.data = new Data();

dataFly.data.Name = dataName;

myEllipse = new Ellipse();

myEllipse.Fill = Cores[i].myEllipse.Fill;

myEllipse.StrokeThickness = 1.5;

myEllipse.Stroke = Brushes.Black;

myEllipse.Width = dataSizeX;

myEllipse.Height = dataSizeY;

Base.Children.Add(myEllipse);

Canvas.SetLeft(myEllipse, Cores[i].Orbits.center\_x + (processesSizeX - 1) / 2);

Canvas.SetTop(myEllipse, Cores[i].Orbits.center\_y + (processesSizeY - 1) / 2);

dataFly.data.myEllipse = myEllipse;

flyingData.Add(dataFly);

}

}

break;

case "da":

dataName = currentCommand[2];

myEllipse = new Ellipse();

mySolidColorBrush = new SolidColorBrush();

mySolidColorBrush.Color = Colors.White;

myEllipse.Fill = mySolidColorBrush;

myEllipse.StrokeThickness = 1.5;

myEllipse.Stroke = Brushes.Black;

myEllipse.Width = 1;

myEllipse.Height = 1;

Data myEllipseData = new Data();

myEllipseData.Name = dataName;

myEllipseData.myEllipse = myEllipse;

int position = (int)Vault.Add(myEllipseData);

Base.Children.Add(Vault.vault[position].myEllipse);

Canvas.SetLeft(Vault.vault[position].myEllipse, getX(Vault.getX(position)));

Canvas.SetTop(Vault.vault[position].myEllipse, getY(Vault.getY(position)));

widthAnimation = new DoubleAnimation();

widthAnimation.To = dataSizeX;

widthAnimation.Duration = TimeSpan.FromSeconds(1);

MainWindow.Vault.vault[position].myEllipse.BeginAnimation(Ellipse.WidthProperty, widthAnimation);

heightAnimation = new DoubleAnimation();

heightAnimation.To = dataSizeY;

heightAnimation.Duration = TimeSpan.FromSeconds(1);

MainWindow.Vault.vault[position].myEllipse.BeginAnimation(Ellipse.HeightProperty, heightAnimation);

break;

case "dd":

dataName = currentCommand[2];

Base.Children.Remove(Vault.get(dataName).myEllipse);

Vault.delete(dataName);

break;

}

}

for (int i = 0; i < Cores.Length; i++)

{

if (Cores[i] != null)

{

alfa[i] = (alfa[i] + alfaV[i]) % (2 \* Math.PI);

Canvas.SetLeft(Cores[i].myEllipse, center\_x - 11 + (100 + 50 \* i) \* Math.Cos(alfa[i]));

Canvas.SetTop(Cores[i].myEllipse, center\_y - 11 + (100 + 50 \* i) \* Math.Sin(alfa[i]));

Cores[i].Orbits.setCenter(Canvas.GetLeft(Cores[i].myEllipse), Canvas.GetTop(Cores[i].myEllipse));

Canvas.SetLeft(Cores[i].Text, Cores[i].Orbits.getTextX());

Canvas.SetTop(Cores[i].Text, Cores[i].Orbits.getTextY());

for (int j = 0; j < Cores[i].Orbits.orbits.Length; j++)

{

for (int k = 0; k < Cores[i].Orbits.orbits[j].Length; k++)

{

if (Cores[i].Orbits.orbits[j][k] != null)

{

Canvas.SetLeft(Cores[i].Orbits.orbits[j][k].myEllipse, Cores[i].Orbits.getX(new orbitPosition(j, k)));

Canvas.SetTop(Cores[i].Orbits.orbits[j][k].myEllipse, Cores[i].Orbits.getY(new orbitPosition(j, k)));

if ((counter - Cores[i].Orbits.addTimes[j][k]) > interval)

{

Base.Children.Remove(Cores[i].Orbits.orbits[j][k].myEllipse);

Cores[i].Orbits.addTimes[j][k] = 0;

Cores[i].Orbits.availableOrbits[j][k] = true;

Cores[i].Orbits.orbits[j][k] = null;

}

}

}

}

}

}

for (int i = 0; i < flyingData.Count; i++)

{

double direction\_x;

double direction\_y;

double length;

switch (flyingData[i].destination)

{

case "orbit":

direction\_x = Cores[flyingData[i].coreNumber].Orbits.getX(flyingData[i].op) - Canvas.GetLeft(flyingData[i].data.myEllipse);

direction\_y = Cores[flyingData[i].coreNumber].Orbits.getY(flyingData[i].op) - Canvas.GetTop(flyingData[i].data.myEllipse);

length = Math.Sqrt(direction\_x \* direction\_x + direction\_y \* direction\_y);

if (flyingData[i].pause < 100)

{

flyingData[i].pause++;

}

else

{

if (length < 1)

{

Cores[flyingData[i].coreNumber].Orbits.orbits[flyingData[i].op.orbit][flyingData[i].op.position] = flyingData[i].data;

Cores[flyingData[i].coreNumber].Orbits.addTimes[flyingData[i].op.orbit][flyingData[i].op.position] = counter;

flyingData.RemoveAt(i);

}

else

{

Canvas.SetLeft(flyingData[i].data.myEllipse, Canvas.GetLeft(flyingData[i].data.myEllipse) + 2 \* direction\_x / length);

Canvas.SetTop(flyingData[i].data.myEllipse, Canvas.GetTop(flyingData[i].data.myEllipse) + 2 \* direction\_y / length);

}

}

break;

case "vault":

direction\_x = getX(Vault.getX(flyingData[i].vaultPosition)) - Canvas.GetLeft(flyingData[i].data.myEllipse);

direction\_y = getY(Vault.getY(flyingData[i].vaultPosition)) - Canvas.GetTop(flyingData[i].data.myEllipse);

length = Math.Sqrt(direction\_x \* direction\_x + direction\_y \* direction\_y);

if (length < 1)

{

Canvas.SetLeft(flyingData[i].data.myEllipse, getX(Vault.getX(flyingData[i].vaultPosition)));

Canvas.SetTop(flyingData[i].data.myEllipse, getY(Vault.getY(flyingData[i].vaultPosition)));

Vault.vault[flyingData[i].vaultPosition] = flyingData[i].data;

flyingData.RemoveAt(i);

}

else

{

Canvas.SetLeft(flyingData[i].data.myEllipse, Canvas.GetLeft(flyingData[i].data.myEllipse) + 2 \* direction\_x / length);

Canvas.SetTop(flyingData[i].data.myEllipse, Canvas.GetTop(flyingData[i].data.myEllipse) + 2 \* direction\_y / length);

}

break;

}

}

}

}

}

Код программы специфичен для программирования визуализации с помощью технологии WPF. Мы не будем подробно его разбирать, однако отметим некоторые моменты. Система координат связана с центром экрана, движения объектов по окружностям просчитываются в полярных координатах. С каждым типом объектов связан отдельный класс, описывающий структуру объекта. При нажатии на кнопку “Старт” запускается обработчик события, который считывает и разбирает входной файл с историей работы программы, рассчитывает все необходимые параметры (в частности размер хранилища и начальное его состояние, формирует очередь команд для отображения работы программы), и запускает визуализацию. Визуализация строится на основе таймера. Каждые 10 миллисекунд запускается функция, которая пересчитывает расположение объектов, а так же проверяет, пришло ли время для запуска очередной команды из файла с историей, и если текущее время выполнения программы превышает время, указанное в команде, то команда выполняется и удаляется из очереди команд.

Заключение

В результате проведенной работы была разработана программа-визуализатор RideVis, которую можно использовать для представления работы параллельных программ, написанных для системы RiDE. В качестве сущностей визуализации выступают хранилище, данные, процессы, количество процессоров в системе. В качестве событий, на которых основывается визуализация, выступают добавление/удаление данных в хранилище, запуск/завершение процессов, чтение/запись данных в хранилище. По количеству отображаемых вокруг процессов прочитанных данных, а так же по состоянию хранилища, можно сделать первые выводы об эффективности работы программ, написанных для системы RiDE, потому как оценка эффективности является одной из основных задач.

На данный момент система RiDE активно развивается, многие заложенные в нее идеи и возможности еще не реализованы, поэтому пока что речь идет лишь о первой версии визуализатора, отображающего лишь базовые функции системы. Дальнейшее развитие системы визуализации может быть связано с реализацией некоторых отладочных функций, разработкой средств для статистического анализа эффективности работы программ для системы RiDE, а так же создание базового инструментария для визуального программирования.

Литература