
**Искусственные общества. 2013-2026**

ISSN 2077-5180

URL - http://artsoc.jes.su



Выпуск 1 Том 15. 2020

# Опыт реализации параллельной пространственно-распределенной агент-ориентированной модели с использованием многоядерной архитектуры

**Бахтизин Альберт Рауфович**
*Центральный экономико-математический институт Российской академии наук*
*Российская Федерация, Москва*

**Макаров Валерий Леонидович**
*Центральный экономико-математический институт Российской академии наук*
*Российская Федерация, Москва*

**Хабриев Булат Рамилович**
*ООО «РТ-Развитие бизнеса»*
*Российская Федерация, Москва*

**Аннотация**

В ходе исследований по определению механизмов обеспечения отказоустойчивости современных высокопроизводительных высоконадежных вычислений, применительно к общественным наукам, нами был модифицирован и апробирован подход к распараллеливанию ресурсоемких агент-ориентированных моделей, обмениваются информацией и имеют пространственную привязку. Этот подход был предложен ранее специалистами из Университета Северной Каролины и Университета Айовы (США). В рамках проведенных экспериментов тестировалась производительность параллельной версии модели в зависимости от двух фундаментальных свойств пространственных интерактивных систем: (1) размер пространства, связанного с количеством распределенных по нему агентов, участвующих во взаимодействиях (определяет общий объем межагентных взаимодействий) и (2) радиус взаимодействия (определяет максимальное расстояние, на котором пара агентов может связываться).

**Ключевые слова:** агентная модель,распараллеливание моделей, распределенная среда,распределение агентов

**Дата публикации:** 13.03.2020

**Источник финансирования:**

Работа выполнена в рамках Программы Президиума РАН «Механизмы обеспечения отказоустойчивости современных высокопроизводительных и высоконадежных вычислений».

**Ссылка для цитирования:**

Бахтизин А. Р., Макаров В. Л., Хабриев Б. Р. Опыт реализации параллельной пространственно-распределенной агент-ориентированной модели с использованием многоядерной архитектуры // Искусственные общества. 2020. Т. 15. Выпуск 1. URL: https://artsoc.jes.su/s207751800008235-7-1/ DOI: 10.18254/S207751800008235-7

1

## Введение

В статье рассмотрен зарубежный опыт построения агентных моделей, реализованных с использованием суперкомпьютерных технологий. Большой обзор разработок в этой области приведен в монографии В.Л. Макарова, А.Р. Бахтизина и Е.Д. Сушко (Макаров, Бахтизин, Сушко, 2016), а здесь мы ознакомим с последними, наиболее заметными работами. Так, в исследовании, проведенном специалистами из университета Салерно (Италия) рассматривается распределенная версия пакета MASON, разработанная в университете Джорджа Мейсона (США) для реализации программного кода агент-ориентированных моделей в распределенной среде – D-MASON. Большинство разработчиков агентных моделей в первую очередь являются специалистами в определенных предметных областях, а их программистские навыки зачастую ограничиваются Знаниями специализированных высокоуровневых пакетов, и в этой связи постоянно растет запрос на системы автоматического распараллеливания программного без его переработки, сильно упрощающие построение высокопроизводительных приложений. Среди особенностей пакета D-MASON – эффективное разделение моделируемого пространства (двух и трехмерного), новый механизм согласованности памяти, интеграция с облачными сервисами. В другой работе – из автономного университета Барселоны разработали инструмент для распараллеливания агент-ориентированных моделей – Care HPS (High Performance Simulation), позволяющий в автоматическом режиме решать задачи распределения выполняемого кода, балансировки вычислительной нагрузки, связи и синхронизации. Мы кратко опишем предлагаемый фреймворк, а также результаты экспериментов.

2 В ходе исследований по определению механизмов обеспечения отказоустойчивости современных высокопроизводительных и высоконадежных вычислений, применительно к общественным наукам, нами был модифицирован и апробирован подход к распараллеливанию ресурсоемких агент-ориентированных

моделей, агенты которых обмениваются информацией и имеют пространственную привязку. Этот подход был предложен ранее специалистами из Университета Северной Каролины и Айовского университета (США). В рамках проведенных экспериментов тестировалась производительность параллельной версии модели в зависимости от двух фундаментальных свойств пространственных интерактивных систем: (1) размер пространства, связанного с количеством распределенных по нему агентов, участвующих во взаимодействиях (определяет общий объем межагентных взаимодействий) и (2) радиус взаимодействия (определяет максимальное расстояние, на котором пара агентов может связываться). Оценка преимуществ использования многоядерных аппаратных систем для реализации пространственно-распределенных агент-ориентированных моделей осуществлялась путем сравнения производительности параллельной версии модели против ее последовательного аналога. Полученные результаты приведены в последней части статьи.

3

## D-MASON: масштабируемая распределенная среда построения агент-ориентированных моделей

Пакет D-MASON позволяет реализовывать агент-ориентированные модели в распределенной среде, увеличивая их производительность, при этом обеспечивая обратную совместимость с базовой средой MASON (Cordasco, Scarano, Spagnuolo, 2018).

4 Работа D-MASON основана на парадигме master/slave (ведущий-ведомый), при использовании которой главное приложение разделяет моделируемое пространство на части и распределяет рабочую нагрузку по ведомым процессам, каждый из которых задействует один или несколько логических процессоров (Logical Processors, LP) в соответствии с их вычислительными возможностями (рис. 1). Ведущий процесс устанавливает однозначную связь между LP и обслуживаемыми ячейками, при которой каждый LP отвечает за:

5
- симуляцию агентов, относящихся к соответствующей ячейке;
- обработку события, связанного с миграцией агентов в другие ячейки;
- связь и синхронизацию между соседними ячейками.

6

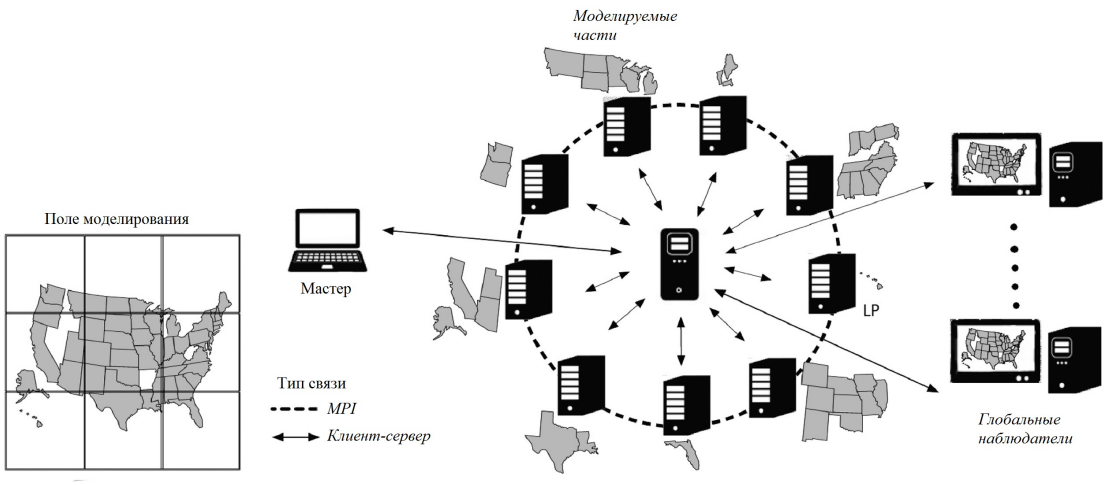


*Рис. 1. Архитектура пакета D-MASON*

7 Основные задачи, решаемые с помощью D-MASON: распределение выполняемой работы, балансировка нагрузки, связь между процессами, синхронизация и воспроизводимость. Ниже рассмотрим их подробнее.

8 ***Распределение работы*** Сложной и неоднозначной проблемой при распараллеливании программного кода является декомпозиция общей задачи на отдельные подзадачи и распределение их по всем LP. В случае реализации агент-ориентированной модели, самым простым способом ее решения является равномерное деление агентов по свободным LP. Такой подход, называемый разработчиками как «разделение агентов» (agents partitioning), хотя и обеспечивает сбалансированную нагрузку на процессоры, но создает проблемы при синхронизации в случае высокой степени связности между агентами и необходимости обмена большим количеством сообщений. Для решения обозначенной проблемы, при разбиении всей совокупности агентов разработчики группируют их в зависимости от набора связей в рамках определенной фиксированной окрестности, называемой «область интересов» (Area of Interest (AOI)) с последующим распределением по LP. Поскольку наиболее интенсивное взаимодействие между агентами происходит в пределах AOI, потери в производительности при синхронизации также снижаются.

9 ***Балансировки нагрузки*** Поскольку агенты могут менять свою группу общения, а также географическое местоположение, разработчики предусмотрели адаптивную технологию перераспределения агентов по LP, обеспечивающую сбалансированность нагрузки по процессорам и эффективность всего приложения, которая будет рассмотрена ниже.

10 ***Связь*** В D-MASON связь между LP основана на поведенческом шаблоне проектирования передачи сообщений «Издатель/Подписчик» (Publish/Subscribe (P/S)), в рамках которого отправители не связаны с подписчиками напрямую, а очереди сообщений делятся на классы, которые не содержат сведения о подписчиках и издателях.

11 Первая версия D-MASON была реализована с использованием стандарта Java Message Service (JMS) и сервера Apache ActiveMQ в качестве провайдера JMS. Ниже будет рассмотрено реализованное разработчиками улучшение шаблона с использованием децентрализованной стратегии на базе интерфейса MPI.

12 ***Синхронизация*** Для корректного выполнения параллельной версии программы по сравнению с ее последовательным аналогом, каждый LP должен непрерывно собирать информацию обо всех связанных с ним ячейках. В D-MASON выполнение программы осуществляется в виде отдельных шагов, каждый из которых состоит из трех этапов: (1) *симуляция*, (2) *связь* и (3) *синхронизация* (рис. 2). В процессе связи LP отправляет своим соседям информацию об агентах, которые мигрируют к ним, а также об агентах, которые могут попасть в AOI соседней ячейки. D-MASON использует механизм консервативной синхронизации для последовательной интеграции распределенных симуляций: сначала локально

синхронизируется информация, полученная LP с соседних ячеек на шаге $t$ – 1, а затем наступает следующий шаг симуляции $t$.

13 Процесс локальной синхронизации в виде последовательной диаграммы отображен на рис. 3, где две соседние ячейки ([0-0] и [0-1]) непрерывно обмениваются сообщениями о мигрирующих агентах и агентах, находящихся вблизи границы.

14

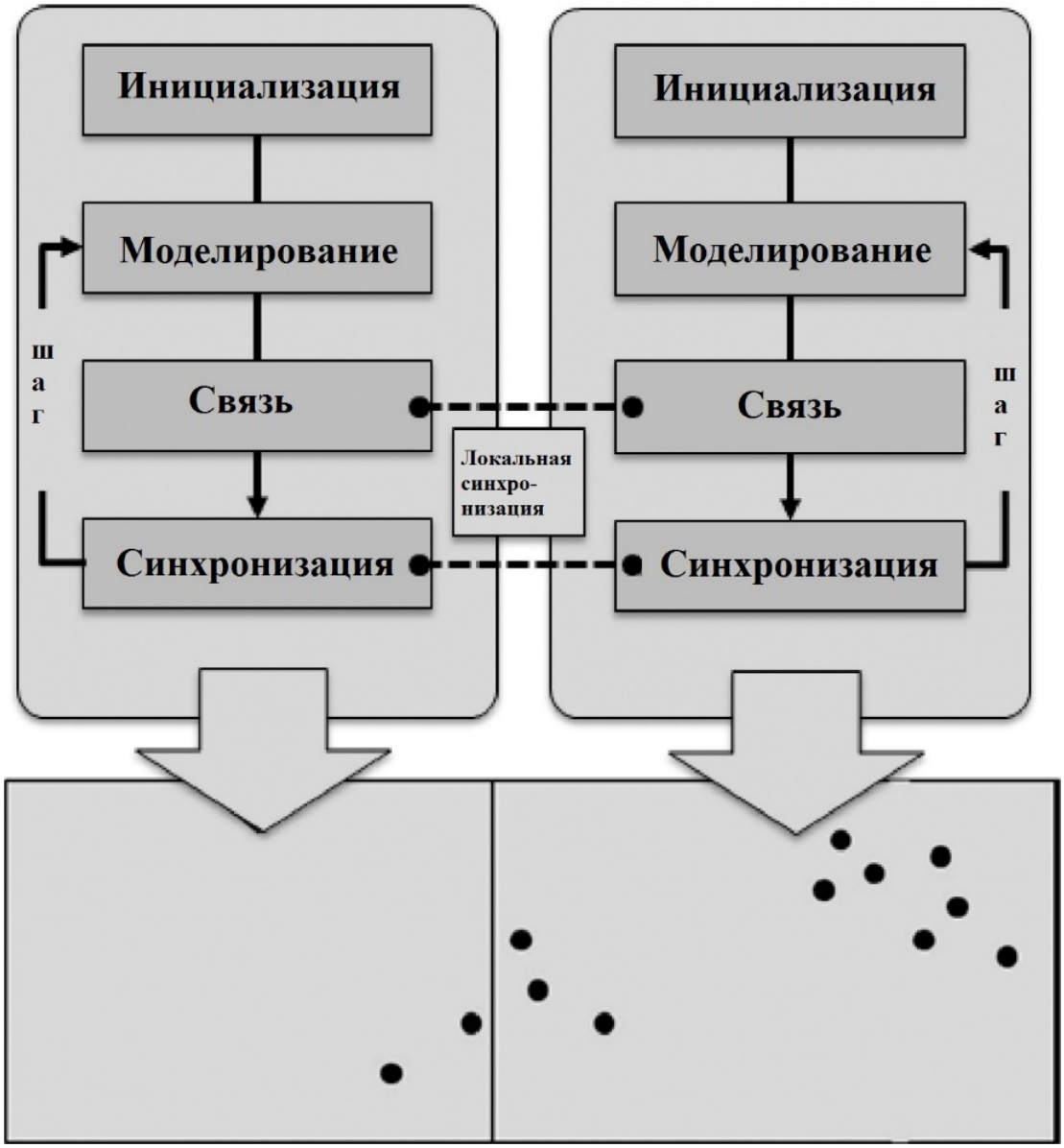



*Рис. 2. Синхронизация процессоров LP в пакете D-MASON: два прямоугольника представляют две соседние ячейки, обслуживаемые разными LP; черные точки – агенты модели. После инициализации, каждый шаг модельного времени состоит из трех этапов: (1) симуляция, когда каждый LP работает самостоятельно, (2) связь и (3) синхронизация (обмен сообщениями между LP)*

15

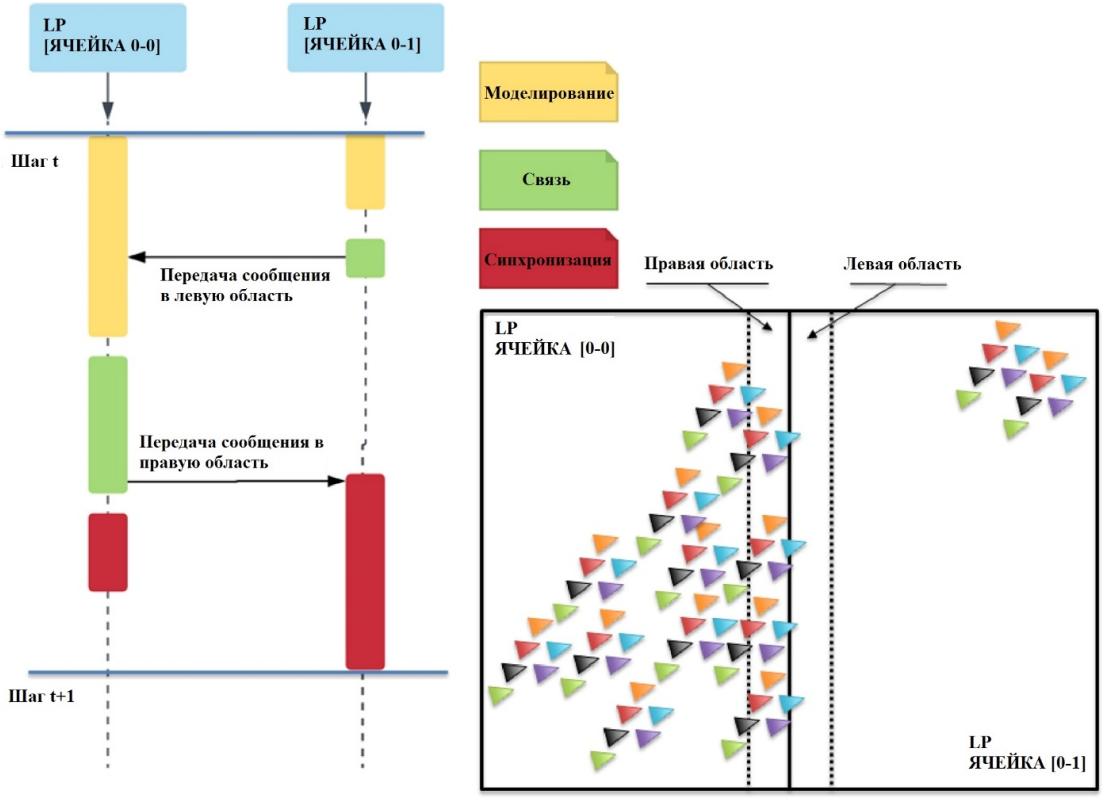



*Рис. 3. Диаграмма локальной синхронизации в пакете D-MASON*

16 ***Воспроизводимость*** Разработчики D-MASON акцентируют особое внимание на то, что порядок, согласно которому вычисляется код агентов, не влияет на результаты симуляций. Таким образом, если реализуемая имитационная модель является полностью детерминированной, то результаты распределенных симуляций с использованием D-MASON идентичны результатам последовательной версии пакета MASON. В том случае, когда симуляции проводятся с использованием случайных начальных значений некоторых параметров, результаты по-прежнему воспроизводимы при условии, что каждая симуляция выполняется с одинаковым количеством LP. Если же число LP меняется, результаты симуляций также разнятся, но и в этом случае тесты показали, что усредненные оценки по множеству прогонов достаточно стабильны.

17

## Архитектура пакета D-MASON

При разработке архитектуры были установлены следующие требования к пакету D-MASON: (1) эффективность использования аппаратной платформы, (2) возможность разработки с использованием различных парадигм моделирования, (3) удобный для пользователя интерфейс, (4) корректность расчетов. Для удовлетворения этим требованиям, D-MASON включает в себя четыре функциональных блока (уровня), отображенных на рис. 4:

18 1. Блок управления системой (*System Management*), предоставляющий удобные средства для регулирования работы симуляций в распределенной системе. 2. Блок визуализации (*Visualization*), позволяющий разработчикам создавать продвинутые средства отображения результатов работы агентных моделей. 3. Блок, обеспечивающий связь (*Communication*) с другими блоками

(уровнями). 4. Блок распределенного моделирования (*Distributed Simulation* (*DS*)), обеспечивающий работу системы на множестве вычислительных узлов.

19 Блок DS является ядром D-MASON и включает в себя два основных пакета: *Engine* и *Field*, первый из которых состоит из трех объектов:

20
- *DistributedState*: обеспечивает идентификацию ячеек и миграцию агентов между ними.
- *DistributedMultiSchedule*: отслеживает время симуляции и процесс синхронизации между этапами симуляции.
- *RemoteAgent*: представляет абстракцию распределенного объекта Steppable.

21


*Рис. 4. Архитектура пакета D-MASON*

22 В листинге 1 приведен простой пример реализации объекта DistributedState. При инициализации для каждого LP задаются 100 агентов и в двухмерном непрерывном пространстве (DContinuousGrid2D) случайным образом определяются их позиции.

23

```
1 public class DSimulation extends DistributedState<Double2D>
2 {
3       public DContinuousGrid2D sim_field;
4       public DSimulation (GeneralParam params, String prefix)
5       {super (params, new DistributedMultiSchedule<Double2D> (), prefix,
                params.getConnectionType ());}
6       public void start ()
7       {super.start ();
8       try {simfield = DContinuousGrid2DFactory.
                createDContinuous2D(10.0/1.5, 200, 200, this, super.AOI,
                TYPE.pos_i, TYPE.pos_j, super.rows, super.columns, MODE,
                "dfield1", topicPrefix, true);
9               init_connection ();
10      } catch (DMasonException e) {e.printStackTrace ();}
11      DAgent agent = null;
12      for (int i = 0; i < 100; i++) {
13              agent=new DAgent (this, new Double2D (0, 0),
                        this.random.nextInt ());
14              agent.setPos (sim_field.getAvailableRandomLocation ());
15              sim_field.setObjectLocation (agent, agent.pos);
16              agent.setColor (Color.RED);
17              schedule.scheduleOnce (sim_field);}
18      }
19      public DistributedField2D getField() {return sim_field;}
20      publi c void addToField (RemotePositionedAgent rm, Double2D loc)
21      {sim_field.setObjectLocation (rm, loc);}
22      public SimState getState() {return this;}
23 }
```

*Листинг 1. DSimulation*

24 В листинге 2 показан код для очень простого агента, создаваемого с помощью двух объектов: абстрактным классом RemoteDAgent и реализующем агента классом – DAgent. Эти два объекта являются подклассами RemoteAgent и определяют местонахождение агентов в пространстве и их положение в рамках сетевой структуры (без привязки к географическим координатам).

25

```
1 // абстрактный базовый класс агента
2 public abstract class RemoteDAgent<E> implements Serializable,
        RemotePositionedAgent<E>  {
3       public E pos; // местоположение агента
4       public String id; // идентификатор удаленного агента
5       public RemoteDAgent  () {}
6       public RemoteDAgent  (DistributedState<E> state) {
7               int i = state.nextId ();
8               this.id = state.getType ().toString () + "–" + i ;
9       }
10      public E getPos () {return pos;}
11      public void s e tPos (E pos ) { this.pos = pos;}
12      public String getId () {return id;}
13      public void setId (String id) { this.id = id;}
14      public boolean equals (Object obj) {// код опущен
15 }
16 // класс агента
17 public class DAgent extends RemoteDAgent<Double2D>{
18      private int val;
19      public DAgent () {} // требуется для сериализации D-MASON
20      public DAgent (String id, Double2D location, Integer val) {
21              this.id = id; this.pos = location; this.val = val;
22      }
23      public Bag getNeighbors (DistributedState<Double2D>  sm) {
24              return ((DContinuousGrid2D)sm.getField ()).
                        getNeighborsExactlyWithinDistance  (pos, 10 , t rue);
25      }
26      public void step (SimState state) {
27              Bag b = getNeighbors ((DistributedState) state);
28              int max=val;
29              for (Object f: b) {
30                      DAgent d = (DAgent) f;
31                      int dval = d.getVal ();
32                      if (max < dval) max = dval;
33              } this.val = max;
34      }
35      public int getVal (DistributedMultiSchedule  schedule){ return val;}
36 }
```

*Листинг 2. DAgent*

26 В листинге 3 приведен пример кода, реализующий экземпляр брокера сообщений Apache ActiveMQ на локальном компьютере. В тестовом примере выполняется 8 LP.

27

```
1 public class DTestLocalMachine  {
2       private static int numSteps = 100, //количество шагов
3       private static int rows = 2; //количество рядов
4       private static int columns = 4; //количество столбцов
5       private static int AOI = 10; //AOI
6       private static int CONNECTION_TYPE=ConnectionType.pureActiveMQ;
7       private static String ip = "127.0.0.1";
8       private static String port = "61616";
9       private static String topicPrefix = " toysim";
10      private static int MODE = DistributedFie1d2D.UNIFORM_PARTITIONING_MODE;
11      public static void main (String [] args) {
12              (new ActiveMQStarter ()).startActivemq (); // запускается локальный ActiveMQ
13              System.setProperty ("org.apache.activemq.SERIALIZABLE_PACKAGES",  "*");
14              class worker extends Thread {
15                      private DistributedState <?> ds;
16                      public worker (DistributedState <?> ds) {this.ds = ds; ds.start ();}
17                      public void run () {
18                              int i = 0;
19                              while ( i!= numSteps) {
20                                      ds. schedule.step(ds);  i++;
21                              } System.exit (0);
22                      }
23              } ArrayList <worker> myWorker = new ArrayList <worker> ();
24              for (int i = 0; i < rows; i++) {
25                      for (int j = 0; j < columns; j++) {
26                              GeneralParam genParam = new GeneralParam (null, null,
                                        AOI, rows, columns, null, MODE, CONNECTION_TYPE);
27                              genParam.setI (i); genParam.setJ (j);
28                              genParam.setIp(ip); genParam.setPort (port);
29                              ArrayList <EntryParam <String, Object>> simParams = new
                                        ArrayList <EntryParam <String, Object >> ();
30                              DSimulation sim = new DSimulation (genParam,
                                        simParams, topicPrefix) ;
31                              worker a = new worker (sim);
32                              myWorker.add (a);
33                      }
34              } for (worker w: myWorker) w.start ();
35      }
36 }
```

*Листинг 3. DTestLocalMachine*

28

## Некоторые новые функции D-MASON

Изначально D-MASON был разработан для использования в слабосвязанной среде с разнородными машинами для их задействования при решении ресурсоемких задач. На данный момент D-MASON может быть развернут в самых различных вычислительных средах – от одного многоядерного компьютера до неограниченного количества машин, доступных в облаке. Помимо этого, в D-MASON реализованы различные рассмотренные ниже механизмы, позволяющие значительно увеличить производительность приложений.

29 ***1. Разделение пространства в D-MASON*** D-MASON поддерживает два вида пространств: геометрическое (двух- или трехмерное) и сетевое, в котором агенты распределяются по узлам сети, а отношения между ними описываются ребрами. Для разделения геометрических пространств предлагается два подхода:

30 • ***Равномерное разбиение***, при котором моделируемое пространство делится на ячейки одинакового размера. На рис. 5 приведен упрощенный пример разделения двухмерного пространства, представляющего США, на 9 ячеек. Желтая, красная и зеленая зоны – это перекрывающиеся области между ячейками AOI, о которых упоминалось выше. К сожалению, такое разбиение не гарантирует сбалансированного разбиения (например, если предположить, что на рис. 5 серые зоны являются территориями с высокой плотностью агентов, то вычислительная нагрузка на LP будет очень неравномерной).

31

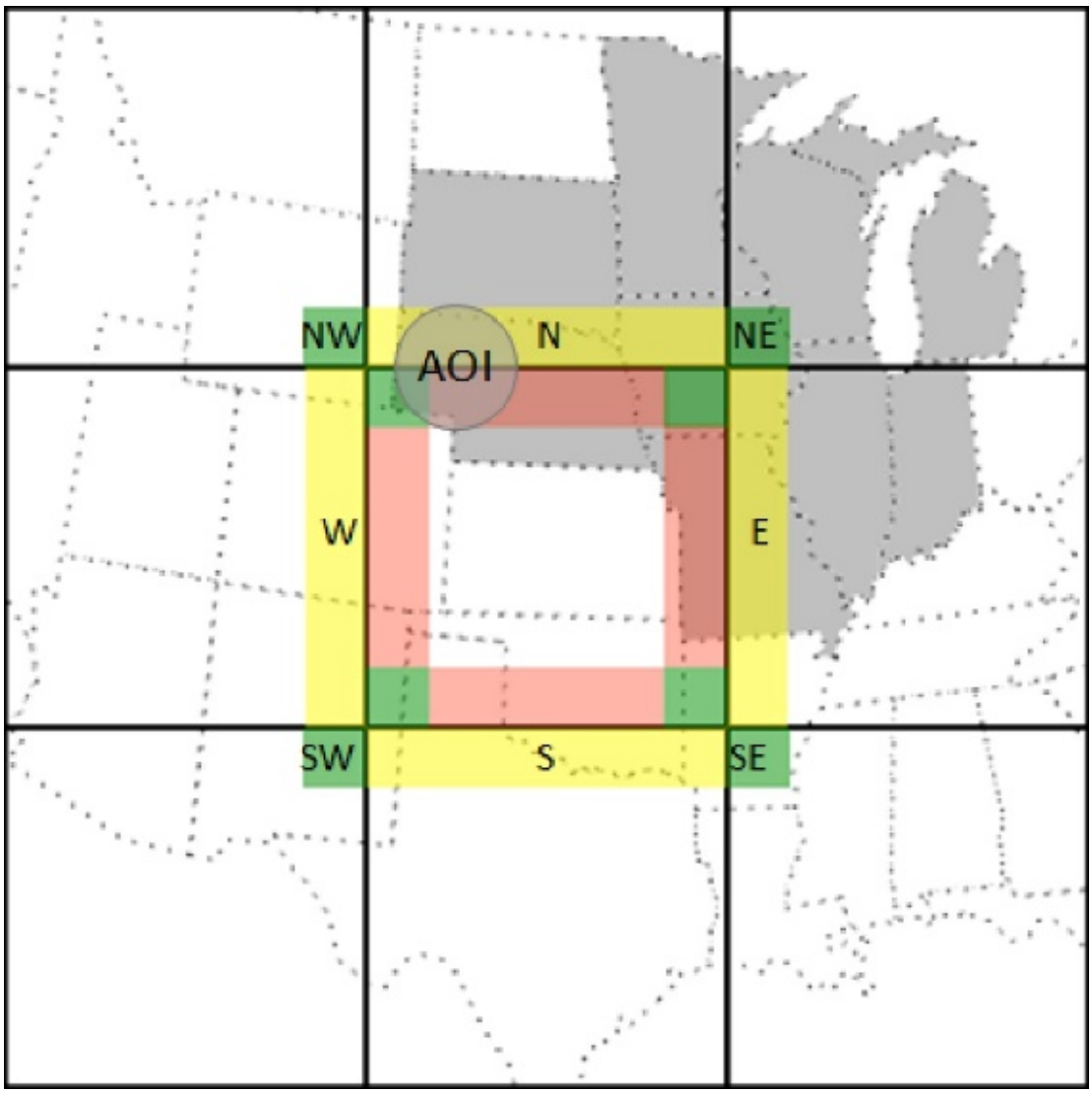



*Рис. 5. Равномерное разбиение пространства на 9 ячеек*

32 • ***Неравномерное разбиение***, при котором учитывается пространственное распределение агентов для последующей балансировки нагрузки между LP. Для разделения применяется алгоритм построения дерева квадрантов или квадродерева (Quad-tree), в котором каждая его часть делится на четыре сбалансированные области (квадраты, прямоугольники), а точка разделения – центр скопления агентов, взвешенный в соответствии с их вычислительной сложностью. Пример такого разбиения для 9 LP приведен на рис. 6. Как видно, размер ячейки обратно пропорционален ее вычислительной сложности. Ниже будут приведены результаты, показывающие, что такая стратегия улучшает масштабируемость и снижает нагрузку, связанную с синхронизацией.

33

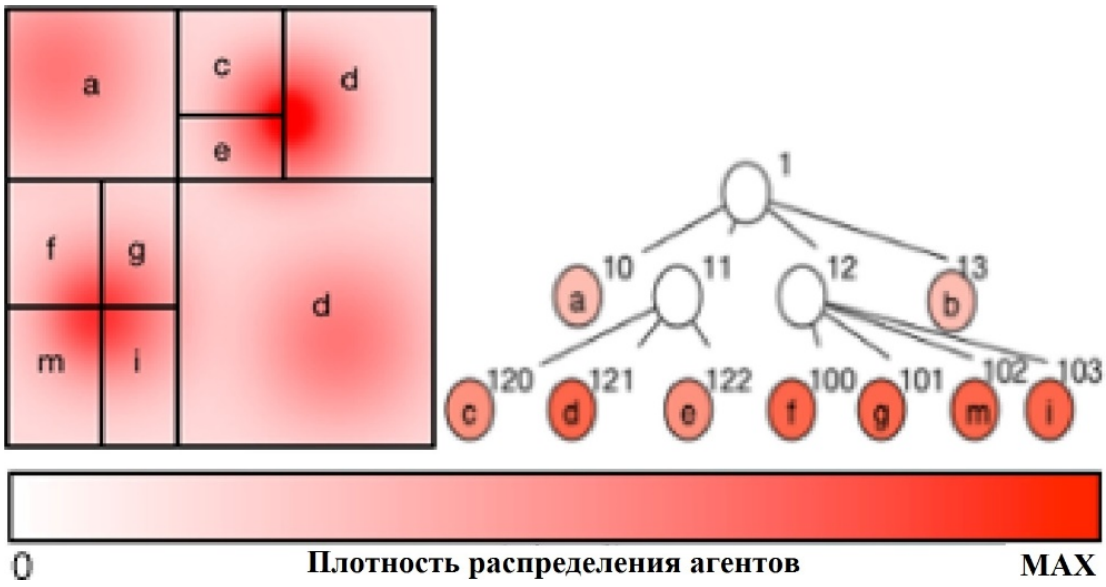



*Рис. 6. Неравномерное разбиение пространства с использованием квадродерева для расчетов на 9 LP. Насыщенность цвета определяется плотностью распределения агентов*

34 Аналогичным образом в D-MASON разбивается трехмерное пространство, но только усложняется процесс синхронизации. Так, для простого примера, приведенного на рис. 5, в случае добавления третьего измерения в прогрессии увеличивается число соседних ячеек. Для 2D случая их 8, а для 3D – 26.

35 ***2. Разделение распределенной сети*** Очень часто изучаемая предметная область, рассматриваемая через призму агентного взаимодействия, может быть структурирована в виде семантической сети. В процессе распараллеливания соответствующий модели, задача состоит в том, чтобы разделить сеть на фиксированный набор компонент таким образом, чтобы они имели приблизительно одинаковый размер, но самое главное – количество соединяющих ее узлы элементов, принадлежащих разным компонентам, должно быть сведено к минимуму (рис. 7). Для решения обозначенной задачи в D-MASON реализован алгоритм графовой декомпозиции METIS.

36

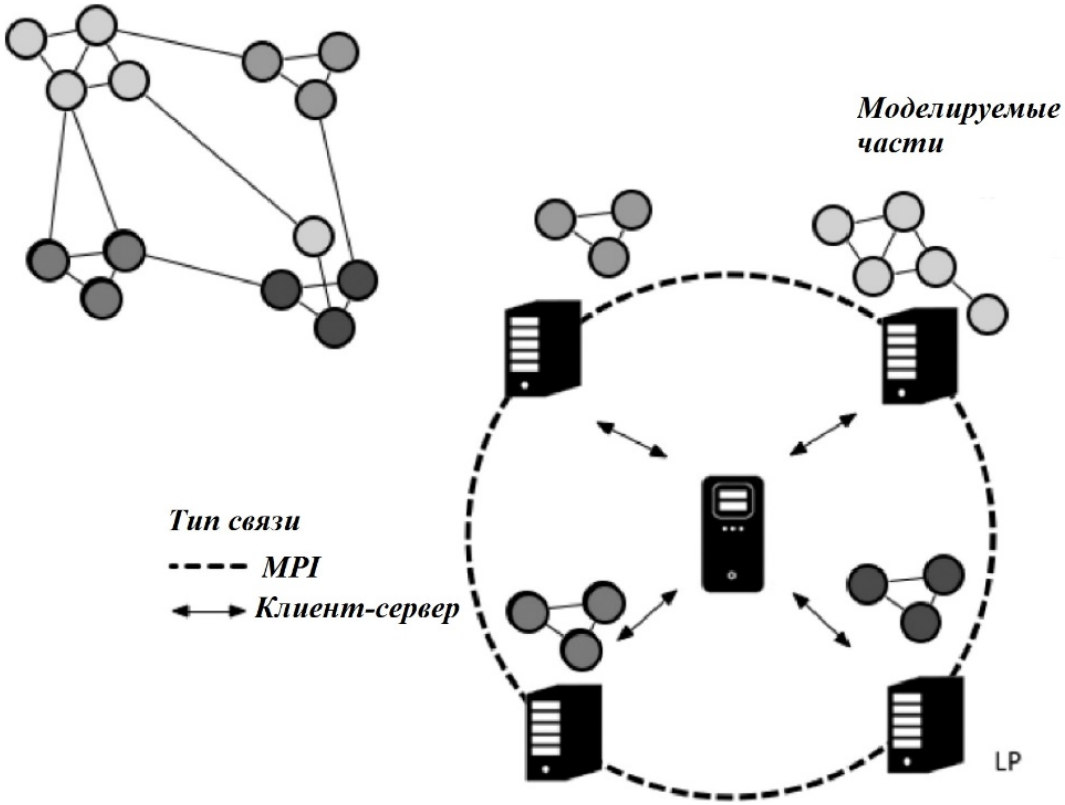



*Рис. 7. Разделение сети в пакете D-MASON*

37 ***3. Механизм согласованности памяти*** Реализация механизма согласованности памяти довольно проста. Во время очередного шага симуляции, при обновлении состояния агента, его предыдущее состояние (полученное в конце предыдущего шага) сохраняется в ассоциативном массиве. При выполнении операции чтения, сначала происходит проверка на наличие в массиве записи о соответствующем агенте. Если запись имеется, то состояние агента считывается, а в противном случае его текущее состояние остается без изменения. По окончании шага симуляции массив очищается. По сравнению с классической двойной буферизацией, данный подход менее требователен к памяти.

38 ***4. Блок, обеспечивающий связь с другими блоками*** В D-MASON предусмотрено два вида связи:

39
- *Централизованная* – используется для архитектур общего назначения с применением JMS и сервера Apache ActiveMQ.
- *Децентрализованная*, которая разработана для экстремально масштабных вычислений (extreme-scale computing) с использованием программного интерфейса MPI.

40 ***5. Управление системой*** Управление D-MASON осуществляется с использованием контейнера сервлетов Jetty, а для пользовательского интерфейса был использован стиль дизайна приложений, разработанный Google. Для пользователя панель управления предоставляет четыре окна со следующими настройками:

41 1. *Окно мониторинга* для отслеживания доступных вычислительных ресурсов в сети, с информацией о количестве узлов, их загруженности, доступной памяти и т.д., с возможностью выбора конкретных ресурсов для запуска планируемой симуляции.
2. *Окно симуляций* со списком всех выполняемых в текущий момент симуляций, позволяющее управлять ими (останавливать, перезапускать, отменять).
3. *Лог системы*, содержащий информацию о всех выполненных симуляциях, их параметрах, сгенерированных файлах.
4. *Окно параметров*, в котором можно менять конфигурация системы (IP адрес, номер порта и др.).

42 ***6. D-MASON в облаке*** В D-MASON реализовано расширение, позволяющее масштабировать агент-ориентированные модели, реализуя среду «Симуляция как услуга» (SIMulation-as-a-Service (SIMaaS)), протестированную на базе облака Amazon Web Services.

43

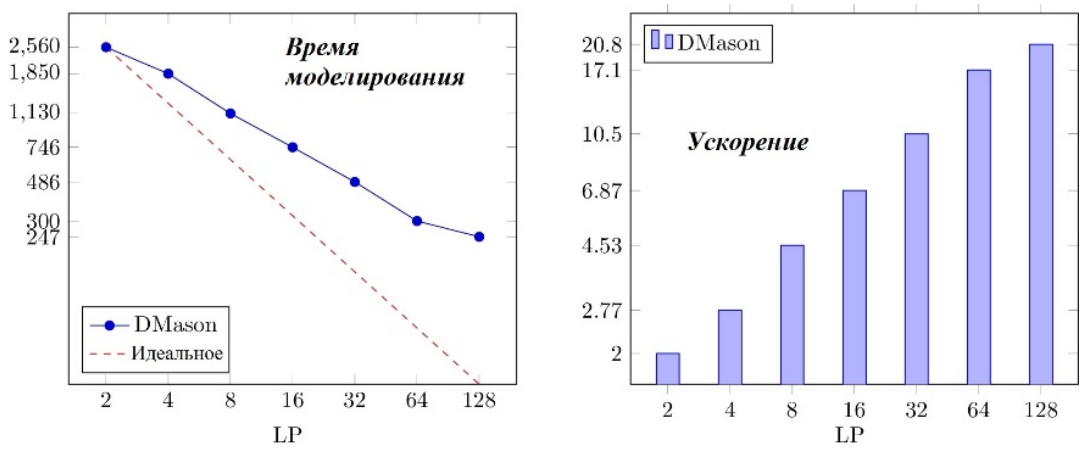
**Вычислительные эксперименты**

В этом разделе будут приведены результаты различных вычислительных экспериментов по оценки масштабируемости D-MASON при реализации классических агент-ориентированных моделей, а также при реализации D-MASON в облачной среде.

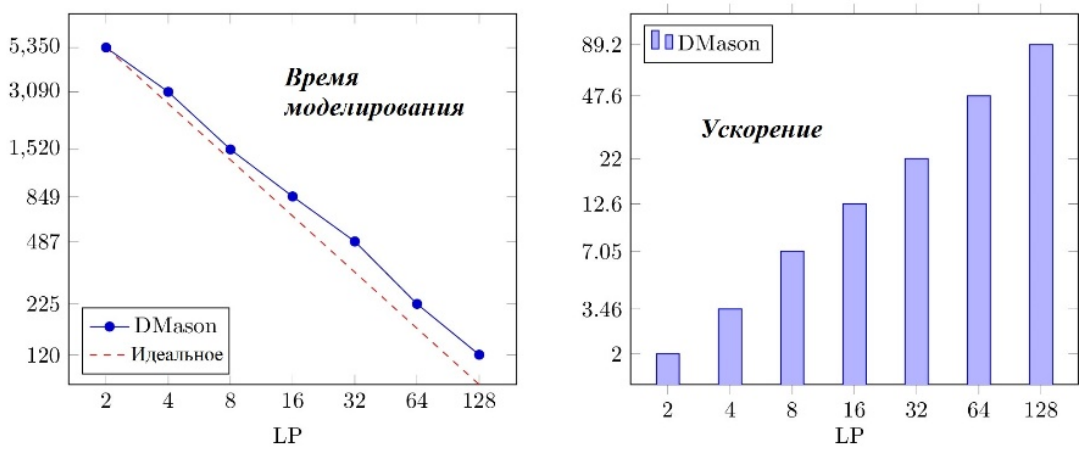
44 Масштабируемость D-MASON для различных агент-ориентированных моделей. В рамках первой серии расчетов оценивалась масштабируемость пакета D-MASON для перечисленных ниже четырех агент-ориентированных моделей, реализованных в 2D-среде:

45 • *Круг* (*Circle*) – модель, в которой рассматривается набор взаимодействующих частиц (притягивающихся и отталкивающихся) в ограниченном пространстве с определенном радиусом. Через некоторое время система приходит в стабильное состояние.
• *Боиды* (*Boids, Flockers*) – модель, разработанная Крейгом Райнольдсом в 1986-м году в качестве симулятора перемещения стаи птиц, в основе которой лежит простой набор правил: сплоченность (особи стараются держаться ближе к центру масс), разделение (особи избегают столкновений), выравнивание (особи двигаются в направлении соседних особей).
• *Игра «Жизнь»* (*Game of Life*) – клеточный автомат, предложенный Джоном Конвеем в 1970 году, где правила перехода весьма просты и зависят от состояний соседних клеток, каждая из которых представляет собой особь, находящуюся в двух состояниях (живая или мертвая).
• *Муравьиный алгоритм* (*Ant colony optimization algorithms*) – модель, основанная на поведении муравьев, которые в процессе поиска питания прокладывают феромонами тропу между своим гнездом и источником пищи.

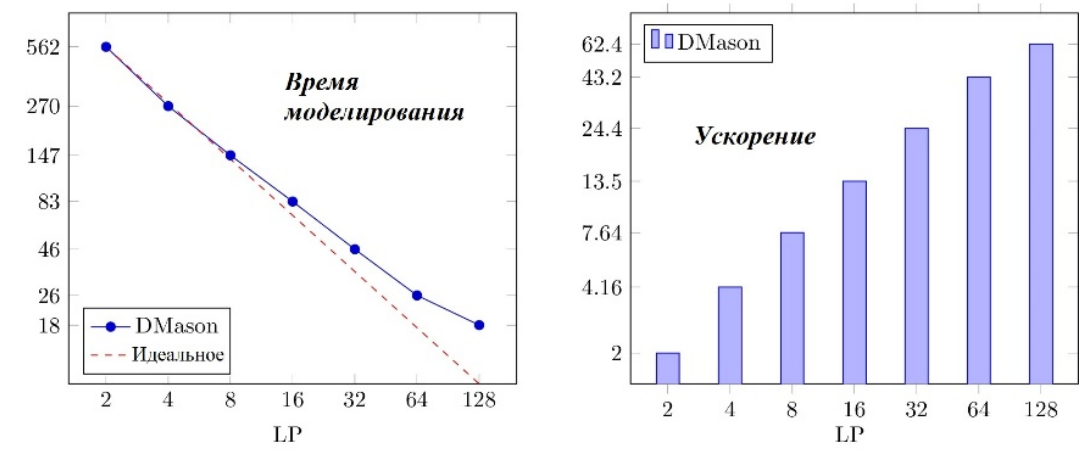
На рис. 8 приведены полученные результаты для каждой из четырех моделей с использованием $10^6$ агентов и 1000 шагов симуляции. Левый график показывает зависимость времени расчетов от числа используемых LP, а график справа – ускорение. Как видно, игра «Жизнь» и модель «Боиды» демонстрируют очень хорошие результаты: ускорение составляет ≈ 62 и ≈ 89 раз соответственно. Наихудшие показатели у муравьиного алгоритма (≈ 2), поскольку распределение муравьев по пространству в ходе симуляций постоянно меняется.




(a) Круг




(b) Боиды




(c) Игра "Жизнь"

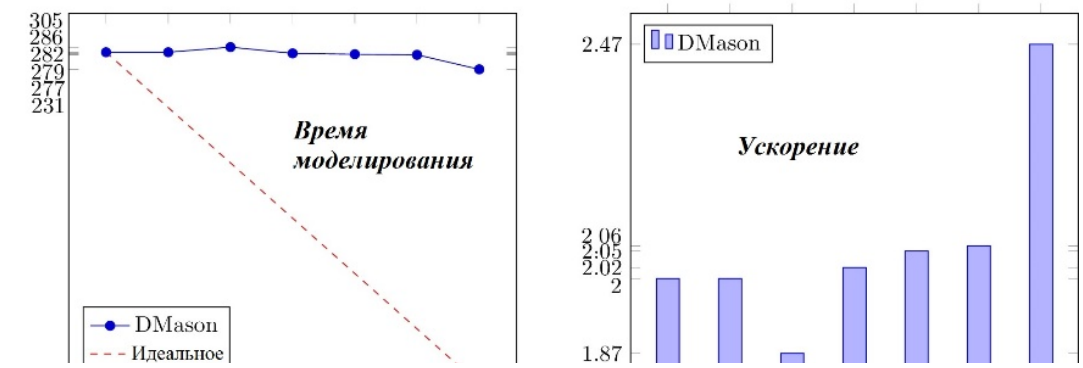


*Рис. 8. Масштабируемость различных агент-ориентированных 2D моделей, реализованных D-MASON для 106 агентов*

48 Помимо расчетов в 2D среде были проведены эксперименты с моделью боидов, реализованной для трехмерного пространства размерностью $10^3 \times 10^3 \times 10^3$ с использованием $10^6$ агентов. На рис. 9 приведены результаты симуляций с применением двух видов связи – централизованной и децентрализованной. Как видно, в обоих случаях ускорение очень хорошее (70,7 раз для централизованной и 72,9 раз для децентрализованной связи).

49

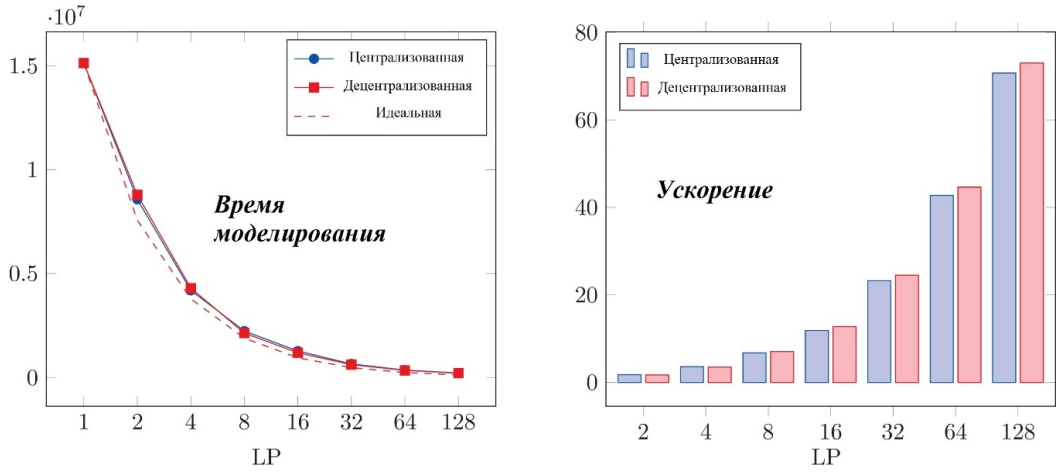



*Рис. 9. Масштабируемость 3D модели в D-MASON для 106 агентов (ось абсцисс – число LP, ось ординат – время симуляций в миллисекундах)*

50 D-MASON в облачной среде: оценка производительности и стоимости. В рамках второй серии расчетов оценивалась производительность D-MASON в облаке с использованием модели боидов, реализованной для 2D пространства размерностью 6400×6400 с $10^6$ агентов. Каждая симуляция выполнялась в течение 15 минут, а в качестве результатов рассматривалось количество выполненных шагов за это время, а также стоимость облачного сервиса. Девять стратегий пространственного разделения (4×4, 4×8, 6×8, 8×8, 10×8, 12×8, 14×8, 16×8, 12×12) соответственно определили 16, 32, 48, 64, 80, 96, 112, 128 и 144 ячеек для эквивалентного количества задействованных LP. Для расчетов был использован веб-сервис, предоставляющий вычислительные мощности в облаке – Amazon Elastic Compute Cloud (Amazon EC2) с инстансами (instance) c4.4xlarge со следующими характеристиками: процессор Intel Xeon E5-2666 v3 (Haswell) с 16 vCPU и 30 ГБ памяти. Стоимость сервиса на момент проведения расчетов составляла 0,80 $ в час (по запросу) или 0,14 $ в час (по свободным ресурсам). Полученные результаты, являющиеся усредненными величинами после 10 симуляций для каждой из 9 стратегий, приведены в таблице 1.

51 Таблица 1. D-MASON в облачной среде: сравнение производительности и стоимости

| Число инстансов | Число LP | Способ разбиения пространства | Количество выполненных шагов симуляции за 15 минут | Общая стоимость сервиса по запросу ($ в час) | Общая стоимость сервиса по свободным ресурсам ($ в час) | Общая стоимость сервиса по запросу ($ за 1000 шагов) |
|---|---|---|---|---|---|---|

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 16 | 4×4 | 720 | 0,80 | 0,14 | 0,28 |
| 2 | 32 | 4×8 | 1251 | 1,59 | 0,27 | 0,32 |
| 3 | 48 | 6×8 | 1784 | 2,39 | 0,41 | 0,33 |
| 4 | 64 | 8×8 | 2849 | 3,18 | 0,54 | 0,28 |
| 5 | 80 | 10×8 | 3817 | 3,98 | 0,68 | 0,26 |
| 6 | 96 | 12×8 | 4795 | 4,78 | 0.81 | 0,25 |
| 7 | 112 | 14×8 | 5806 | 5,57 | 0,95 | 0,24 |
| 8 | 128 | 16×8 | 6645 | 6,37 | 1,08 | 0,24 |
| 9 | 144 | 12×12 | 7292 | 7,16 | 1,22 | 0,25 |

52 Как видно, D-MASON в облаке хорошо масштабируется, а затраты на проведение симуляций незначительны для всех рассмотренных случаев.

53

## 2. Care HPS: высокопроизводительный инструмент для параллельного и распределенного агентного моделирования

Первая версия Care HPS появилась в 2004 г. как результат развития агент-ориентированной модели для изучения поведения косяков рыб, построенной с использованием технологии MPI и консервативного алгоритма синхронизации. За период 2005-2008 гг. была улучшена масштабируемость приложения, что позволило значительно увеличить число агентов при запуске симуляций. В 2009 г. в Care HPS для выборочной спецификации агентов был реализован механизм нечеткой логики, в 2010-2012 гг. улучшены механизмы распределения выполняемого кода и балансировки вычислительной нагрузки, а в 2013 г. функционал системы был расширен за счет использования массовой синхронной параллели (Bulk Synchronous Parallel). В 2014 г. фреймворк был существенно модернизирован на уровне реализующих агентов классов с целью лучшей масштабируемости в связи с ростом ресурсоемкости задач. Масштабируемость, безусловно, зависит от конкретной задачи, поэтому в Care HPS предлагаются различные механизмы синхронизации (Borges, Gutierrez-Milla, Luque, Suppi, 2017).

54

### 2.1. Архитектура Case HPS

Care HPS поддерживает как интерфейс передачи сообщений MPI, так и технологию OpenMP и содержит в себе несколько компонент, реализованных на языке C++. Пользователи решают задачу проектирования модели (в том числе с использованием готовых функциональных элементов управления), а всю работу по распределению агентов по процессорам, синхронизации процессов и т.д. выполняет фреймворк. Далее рассмотрим наиболее важные компоненты Care HPS.

55 ***Компоненты агентной модели*** Компонент *Agent* имеет встроенный класс *agent*, реализующий поведение агента в двух- и трехмерном пространствах с использованием следующих полей:

56 1. *position* – определяет положение агента в пространстве;
2. *velocity* – определяет направление агента;
3. *covering radius* – определяет радиус взаимодействия агента с окружающей средой и другими агентами;

4. *member* – используется для сериализации состояния агента (сериализуемый компонент);
5. *ID* – уникальный идентификатор агента;
6. *random_move* – определяет, перемещается ли агент случайно;
7. *alive* – определяет, жив ли агент.

57 Моделируемая задача может потребовать дополнительных полей, которые должны быть реализованы внутри базового метода *update_agent*, выполняемого каждую единицу времени с момента начала симуляции и до ее завершения. Важным функционалом класса *agent* является возможность сериализации состояния агента, осуществляемой с использованием компонента *member*. Это необходимо, поскольку агенты могут мигрировать из одного процесса в другой, с выполнением соответствующих операций удаления и создания, но при этом состояния агентов должны сохраняться.

58 Компонент *Environment* позволяет моделировать окружающее агентов пространство внутри модели, а соответствующий программный класс, являющийся контейнером, содержит методы для взаимодействия с агентами модели. Часто при запуске симуляций требуется сформировать уникальный набор начальных значений для агентов модели. С использованием компонента *Random number generation* можно определить их посредством нормального, экспоненциального, гамма и равномерного распределений. Последний компонент *Model* позволяет производить общую настройку симуляции.

59 ***Компоненты, отвечающие за параллельные вычисления и распределение программного кода*** На этом уровне присутствуют следующие компоненты:

60
- *Logical process* – создает распределенные процессы, связанные посредством передачи сообщений, а также контролирует ход их выполнения.
- *Sync* – осуществляет синхронизацию.
- *Partitioning* – определяет наиболее эффективный механизм распределения агентов по процессам и реализует его.
- *Balancing* – осуществляет балансировку вычислительной нагрузки (назначение агентов для конкретных ядер, изменение размера разделов и т.д.).

61 Разработчики Care HPS сравнили свой продукт с некоторыми наиболее известными аналогами – Flame, D-Mason, Pandora и Repast HPC. Каждый из этих программных средств имеет сильные и слабые стороны, связанные с первоначальными целями их создания. В Care HPS агенты реализуются путем расширения базового класса *Agent*, в пакете Flame посредством настройки X-машины, в D-Mason с помощью расширения абстрактного класса *RemoteAgent*, также позволяющим сериализовать агентов. В Pandora используется общий базовый класс, посредством которого моделируется любая сущность, а в RepastHPC пользователь должен реализовать функционал агентов с помощью методов базового класса и более того, каждый агент должен реализовать интерфейс *Agent*.

62 Для синхронизации в Care HPS используется метод обратного вызова, во Flame – «доска сообщений», гарантирующая, что все агенты будут видеть один и тот же набор сообщений, в D-Mason применяется пошаговая локальная синхронизация, в Pandora – система планирования, а в RepastHPC – дискретный планировщик событий.

63 Care HPS предлагает три стратегии распределения агентов: кластерное, чистое MPI и гибридное, а для экспертов в области высокопроизводительных вычислений возможна реализация других алгоритмов. Во Flame применяется алгоритм распределения нагрузки с использованием перебора и упорядочения агентов по круговому циклу (Round Robin Partitioning). В D-MASON главное приложение разделяет моделируемое пространство на части и распределяет рабочую нагрузку по ведомым процессам, каждый из которых задействует один или несколько логических процессоров в соответствии с их вычислительными возможностями. В Pandora реализовано пространственное разделение, при котором каждый вычислительный узел получает часть моделируемой среды и агентов, расположенных в его границах. Затем каждая часть делится на четыре раздела, обрабатываемых последовательно. RepastHPC реализует разбиение с помощью концепции проекции, в рамках которой пользователь может выбрать один из трех ее типов: решетка, непрерывное пространство и сеть.

64 Связь между процессами в Care HPS реализуется в следующие моменты: (1) миграция агентов между процессами; (2) синхронизация; (3) инициализация модели; (4) балансировка нагрузки. В свою очередь во Flame связь осуществляется через «доску сообщений», а возможности прямого обмена сообщениями между агентами не существует. В D-MASON связь между LP основана на поведенческом шаблоне проектирования передачи сообщений «Издатель/Подписчик» (Publish/Subscribe (P/S)), в рамках которого отправители не связаны с подписчиками напрямую, а очереди сообщений делятся на классы, которые не содержат сведения о подписчиках и издателях. Pandora использует OpenMP и MPI таким образом, что локальные взаимодействия осуществляются посредством интерфейса OpenMP, а на каждом шаге соседние узлы обмениваются информацией посредством директив MPI.

65 Приведем два самых простых примера использования Care HPS: (1) агент-ориентированная модель перемещения рыб с препятствиями; (2) расширение возможностей пакета путем использования новой стратегии распределения нагрузки.

66 Движение рыбы в модели зависит от ее текущего положения и наличия препятствий, а также от поведения соседних рыб. Код в листинге 4 проверяет, происходит ли столкновение, и если да, то агент отталкивается и меняет направление движения.

67

```
void fish :: check_environment (void) {
vector <object*> obj_env = ENV -> getObjects ();
for (vector <object* > :: iterator ob = obj_env.begin ();
        ob!=obj_env.end ();
        ob++)
if ((*ob) -> check_collision (this -> get_position (),
                              this -> get_velocity (),
                              MAXIMUM_VISION_RANGE))
        this -> repulsion (*this);
}
```

*Листинг 4. Проверка наличия препятствий*

68 Рассмотрим также пример разработки новой стратегии распределения вычислительной нагрузки. На первом этапе создается начальное количество потоков следующим образом: threads = number_agent / total_number_agents ∗ number_cores, где number_agent – количество агентов внутри раздела, total_number_agents – общее количество моделируемых агентов, а number_cores – количество процессов MPI, поделенное на 4 (выбор числа 4 связан с архитектурой используемых ЦП). На втором этапе потоки создаются динамически, и пример такого автоматического распараллеливания приведен в листинге 5.

69

```
void partitionig_strip_hybrid :: execute () {
        long thread = number_agent / total_number_agents*number_cores;
        if (abs (num_thread)>=1){
                if (abs (thread)==1)
                        omp_set_num_threads (2);
                else if ((abs(thread)>= 2) and (abs(thread)<= 8))
                        omp_set_num_threads (abs (thread));
                else if (abs(thread)> 8)
                        omp_set_num_threads (8);
                #pragma omp parallel default(none) firstprivate(i)
                        shared(_bucket)
                {
                        unsigned long j ;
                        #pragma omp for private (j) schedule (dynamic) nowait
                        // для каждого агента выполняется следующее
                        for (j =0; j < _bucket -> size (); j++)
                                _bucket->at (j) -> update_agent();
                }
        }
        else {
        // выполнить цикл
        }
};
```

*Листинг 5. Реализация стратегии распределения нагрузки*

70

## 2.2. Результаты вычислительных экспериментов

Ниже описываются результаты четырех экспериментов, проведенных с тремя агент-ориентированными моделями: (1) покупателей, (2) перемещения рыб и (3) муравьиной колонии.

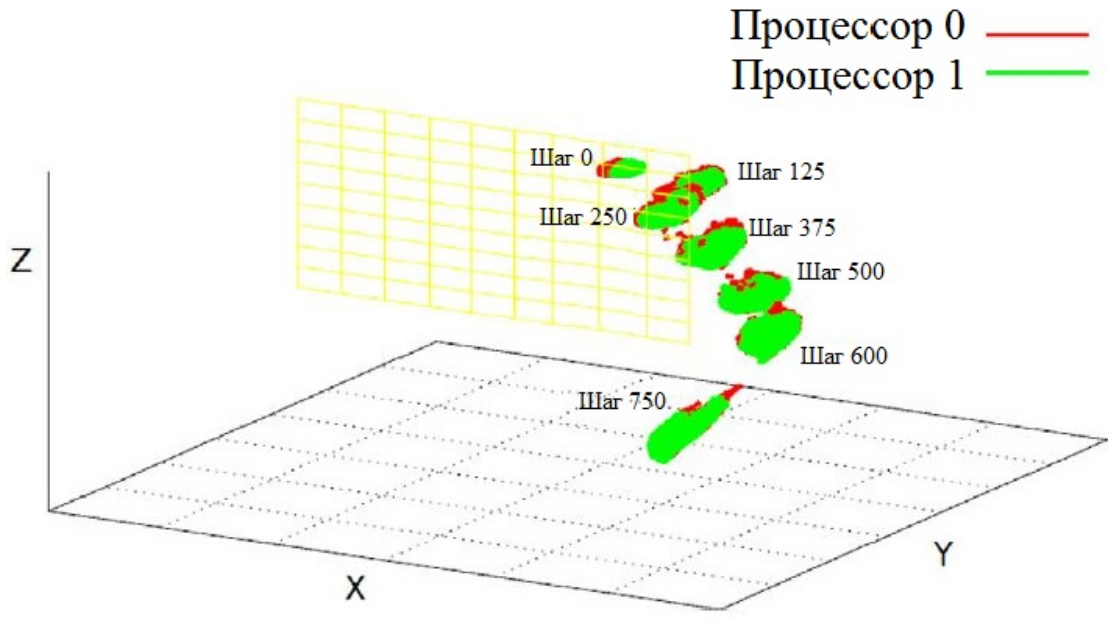
71 ***Первый эксперимент*** В реализованной модели у агента есть список необходимых к покупке товаров, которые распределены по множеству магазинов. Каждый из магазинов продает только один вид товара, а цель агента – купить весь перечень из списка. В рамках расчетов было реализовано три сценария: 1) агенты перемещаются случайным образом, чтобы найти магазин для покупки товара из списка, 2) агенты могут видеть магазины в пределах определенного диапазона, 3) некоторые агенты являются «умными», т.е. вычисляют оптимальный маршрут и более того, агенты модели обмениваются информацией об известных им магазинах.

72 В таблице 2 приведены результаты симуляций в сравнении с версией модели с идентичными параметрами, но реализованной в пакете Netlogo. Как видно, применение алгоритмов Care HPS в случае интенсивного обмена информацией более эффективно.

73 Таблица 2. Сравнение производительности двух реализаций одной модели для трех сценариев (цифры означают количество шагов симуляций для достижения поставленной цели)

| | ***Сценарий 1*** | ***Сценарий 2*** | ***Сценарий 3*** |
|---|---|---|---|
| Netlogo | 14310 | 6983 | 2000 |
| Care HPS | 19082 | 1361 | 348 |

74




*Второй эксперимент В этом эксперименте имитировалась самоорганизация косяка рыб, возникающая в результате локальных взаимодействий отдельных особей, поведение которых зависит от действий соседей. Ни рис. 10 показан результат симуляций для 8192 агентов-рыб с использованием двух процессоров. Трехмерное пространство представляет собой море, а желтая решетка – препятствие, изменившее траекторию движения косяка,*

*который через последовательность шагов проплыл мимо преграды. Рисунок 10. Поведение косяка рыб, избегающих столкновения*

75 ***Третий эксперимент*** С использованием модели муравьиной колонии были проведены расчеты для 1000, 2500, 5000 и 10000 агентов с применением двух стратегий разделения вычислительной нагрузки: чистого разделения с использованием директив MPI и гибридного. На рис. 11 приведены результаты для обеих стратегий, из которых следует, что гибридный механизм существенно эффективнее.

76

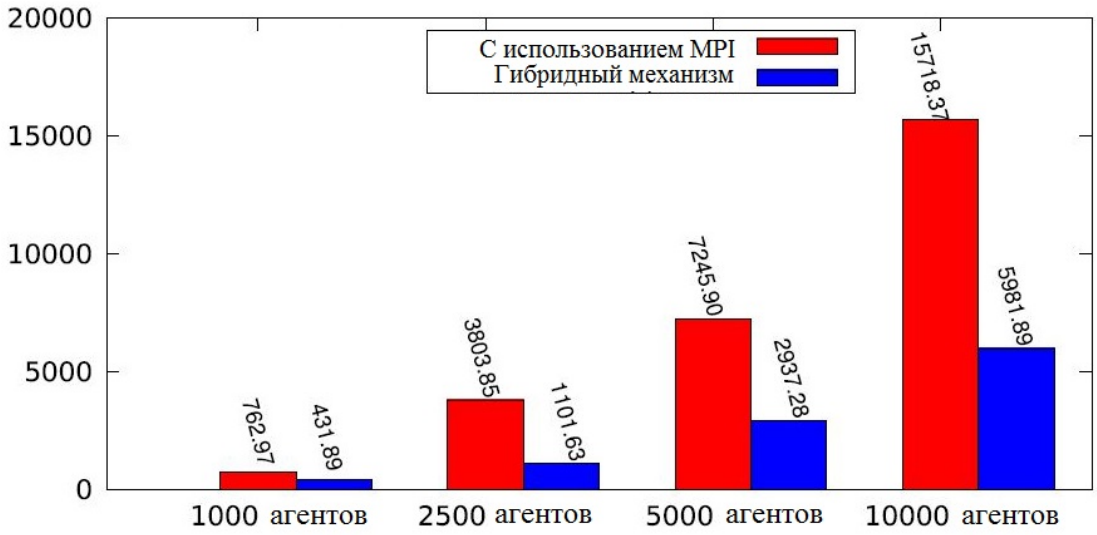



*Рисунок 11. Общее время выполнения алгоритма для чистого MPI и гибридного разделения нагрузки*

77 ***Четвертый эксперимент*** В рамках последнего эксперимента оценивалась масштабируемость агент-ориентированной модели покупателей, реализующей третий сценарий первого эксперимента. Ни рис. 12 и 13 приведены результаты для 10000 и 50000 агентов соответственно, которые осуществляют 50000 шагов симуляции и должны приобрести 10 товаров. Как видно, масштабируемость приложения весьма хорошая.

78

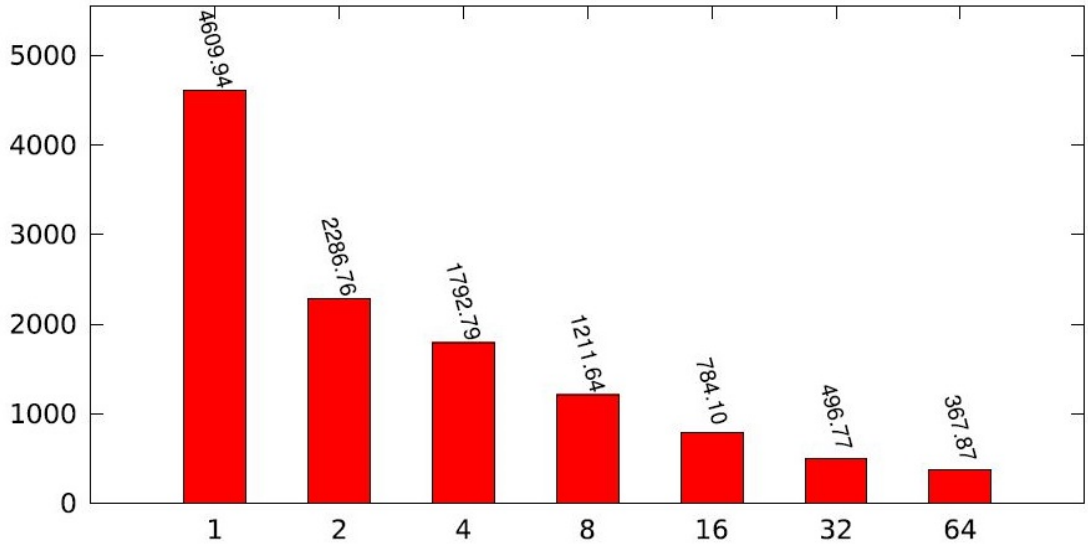


*Рисунок 12. Масштабируемость модели с 10000 агентами, выполнивших 50000 шагов (ось абсцисс – число процессоров, ось ординат – время в секундах)*

79

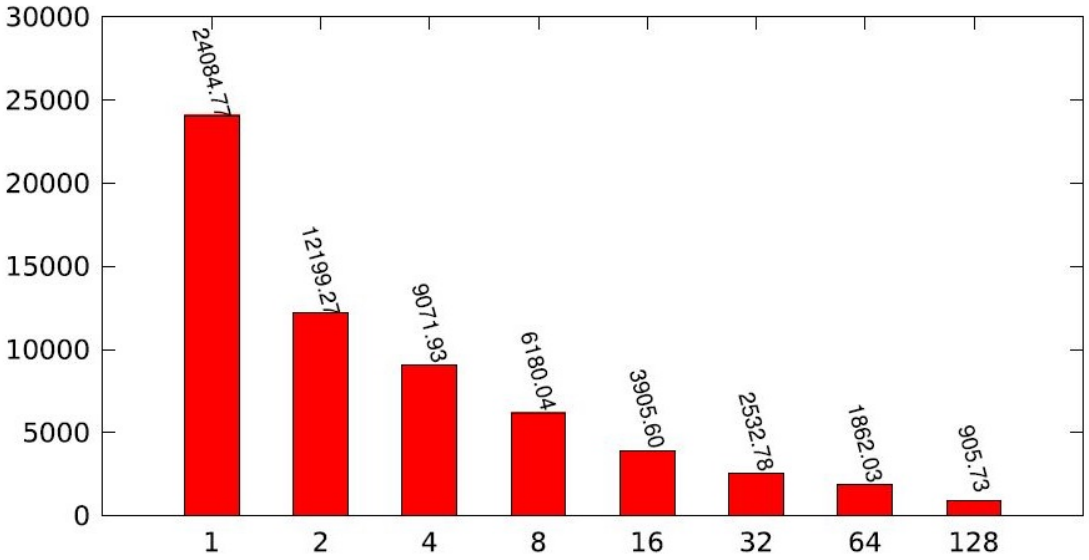



*Рисунок 13. Масштабируемость модели с 50000 агентами, выполнивших 50000 шагов (ось абсцисс – число процессоров, ось ординат – время в секундах)*

80 Далее тестировалась способность Care HPS эффективно работать с моделями большой размерности. На рис. 14 и 15 видно, что для 200000 и 250000 масштабируемость также приемлемая.

81

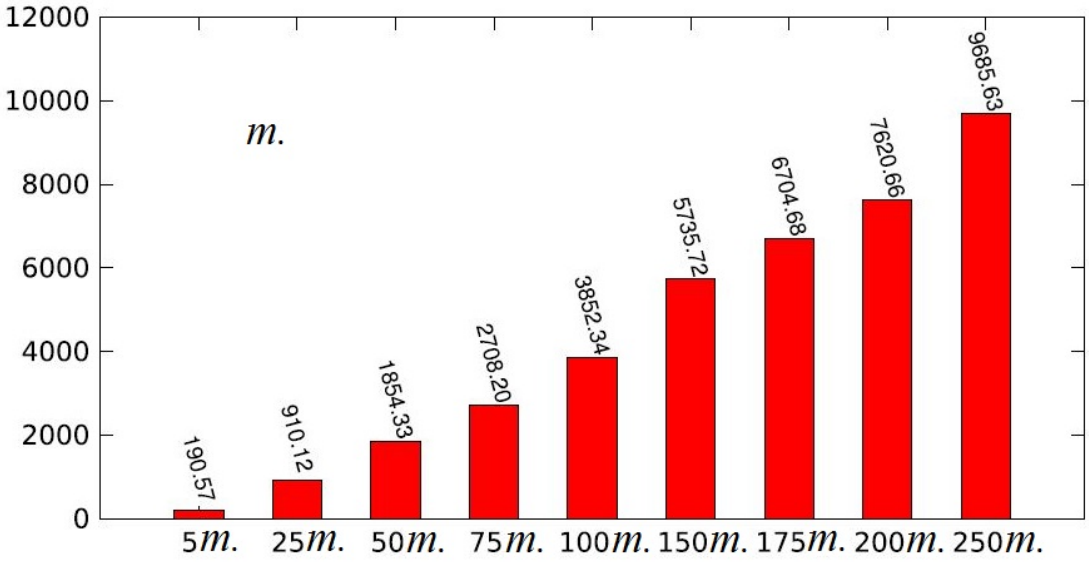



*Рисунок 14. Масштабируемость модели для 50000 шагов, выполненных с использованием 64 ядер (ось абсцисс – число агентов, ось ординат – время в секундах)*

82

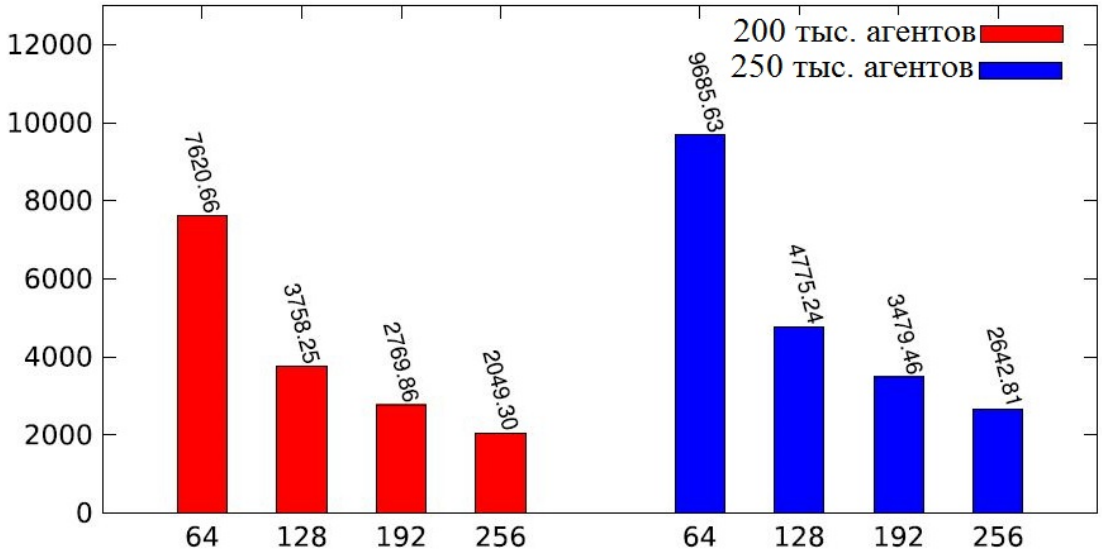



*Рисунок 15. Масштабируемость модели для 200000, 250000 агентов и 50000 шагов, выполненных с использованием 64, 128, 192 и 256 ядер (ось абсцисс, ось ординат – время в секундах)*

83 В настоящее время разработчики Care HPS используют этот фреймворк в проекте, направленном на прогноз распространения лихорадки денге.

84

## 3. Реализация параллельной пространственно-распределенной агент-ориентированной модели с использованием многоядерной архитектуры

В рамках исследования нами был модифицирован и апробирован подход к распараллеливанию ресурсоемких агент-ориентированных моделей, агенты которых обмениваются информацией и имеют пространственную привязку. Этот подход был предложен ранее специалистами из Университета Северной Каролины и Айовского университета (Gong, Tang, Bennett, Thill, 2013) и частично модифицирован в процессе текущего исследования. В процессе проведения экспериментов тестировалась производительность параллельной версии модели в зависимости от двух фундаментальных свойств пространственных интерактивных систем: (1) размер пространства, связанного с количеством распределенных по нему агентов, участвующих во взаимодействиях (определяет общий объем межагентных взаимодействий) и (2) радиус взаимодействия (определяет максимальное расстояние, на котором пара агентов может связываться). Оценка преимуществ использования многоядерных аппаратных систем для реализации пространственно-распределенных агент-ориентированных моделей осуществлялась путем сравнения производительности параллельной версии модели против ее последовательного аналога.

85 В построенной модели агенты взаимодействуют друг с другом (путем обмена сообщениями) в рамках пространства и в процессе взаимодействия приобретают новые знания. Пространство представляет собой двухмерную решетку, в каждой ячейке которой может находиться произвольное число агентов. На каждом шаге симуляции любой агент имеет возможность инициировать общение с только одним другим агентом, в то время как вызываемый агент может

участвовать в нескольких взаимодействиях, инициируемых соответствующими агентами. Таким образом, агент модели на каждом шаге потенциально может иметь несколько экземпляров взаимодействий. В свою очередь взаимодействия между агентами зависят от их пространственного расположения, т.е. конкретный агент может взаимодействовать с другими в окрестности с определенным радиусом. Вероятность того что агент *i* выберет для взаимодействия агента *j* определяется следующей функцией:

86 $$p_{ij} = d_{i,j}^{-1/\alpha}, \quad (1)$$

87 где $d_{i,j}$ – расстояние между двумя агентами, а $\alpha$ – коэффициент, влияющий на вероятность выбора соседствующего агента для взаимодействия. Чем больше значения этого коэффициента, тем шире диапазон взаимодействия (таблица 3).

88 Таблица 3 Зависимость между коэффициентом $\alpha$ и диапазоном взаимодействия (вычислено на основе соотношения (1) для вероятности равной 0.01)

| Коэффициент $\alpha$ | Диапазон взаимодействия (количество ячеек) |
|---|---|
| 0.1 | 2 |
| 0.2 | 3 |
| 0.3 | 4 |
| 0.4 | 6 |
| 0.5 | 10 |
| 0.6 | 16 |
| 0.7 | 25 |
| 0.8 | 40 |
| 0.9 | 63 |
| 1.0 | 100 |
| 1.1 | 158 |
| 1.2 | 251 |
| 1.3 | 398 |
| 1.4 | 631 |
| 1.5 | 1000 |

89 После того как для агента будет рассчитан диапазон взаимодействий, запускается стохастический процесс определения направления для выбора другого агента для общения:

90 $$\theta = U0,360, \quad (2)$$

91 где $\theta$ – параметр, определяющий направление, а $U$ – функция, генерирующая случайное число.

92 Для текущего агента *i* местоположение агента *j*, выбранного для общения вычисляется по следующей формуле:

93 $$\begin{aligned} A_{g_j,r} &= A_{g_i,r} + d_{i,j} \cdot \cos\theta \\ A_{g_j,c} &= A_{g_i,c} + d_{i,j} \cdot \sin\theta \end{aligned} \quad (3)$$

94 где $A_{g_j,r}$ ( $A_{g_i,r}$ ) – номер ряда для агента *i*(*j*), $A_{g_j,c}$ ( $A_{g_i,c}$ ) –столбец для агента *i*(*j*), а . – функция округления.

95 Далее опишем правила общения между агентами, которое возможно при наличии общих интересов. Также отметим, что обучение друг у друга агентов невозможно, если они не имеют общих интересов и имеют противоположные мнения. С другой стороны, если мнения двух агентов достаточно близки, они могут взаимодействовать, и в рамках этого процесса приобретать новые знания. В модели предполагается, что у каждого агента есть порог $\lambda_i$ , определяющий возможность восприятия чужого мнения (соотношение 4). Если численные значения мнения коммуникационного партнера позволяют находиться в пределах установленного порога, то текущий агент меняет свое мнение со скоростью усвоения новой информации $\mu_i$ , равной скорости обучения со стороны партнера.

96
$$O_i^{'} = \begin{cases} 1 - \mu_i \cdot O_i + \mu_i \cdot O_j & \text{если } O_i - O_j \le \lambda_i \\ O_i & \text{в противном случае} \end{cases} \quad (4)$$

97 где $O_i, O_j$ – текущие мнения агентов *i* и *j*; $O_i^{'}$ – измененное мнение после процесса обмена информацией; $\mu_i$ – скорость обучения агента *i*; $\lambda_i$ – порог возможности восприятия стороннего мнения.

98 По сути, соотношение 4 определяет, будет ли взаимодействие между парой агентов или нет. Далее будем считать взаимодействие *успешным*, если взаимодействие состоялось и *неудачным* в противном случае. С точки зрения вычислительного процесса, успешные взаимодействия гораздо более ресурсоемкие по сравнению с неудачными, поскольку первые сопровождаются процедурой обмена мнениями, а вторые – нет. На рис. 16 показана траектория доли неудачных взаимодействий от их общего числа для первых 100 итераций типичного выполнения последовательной версии модели.

99

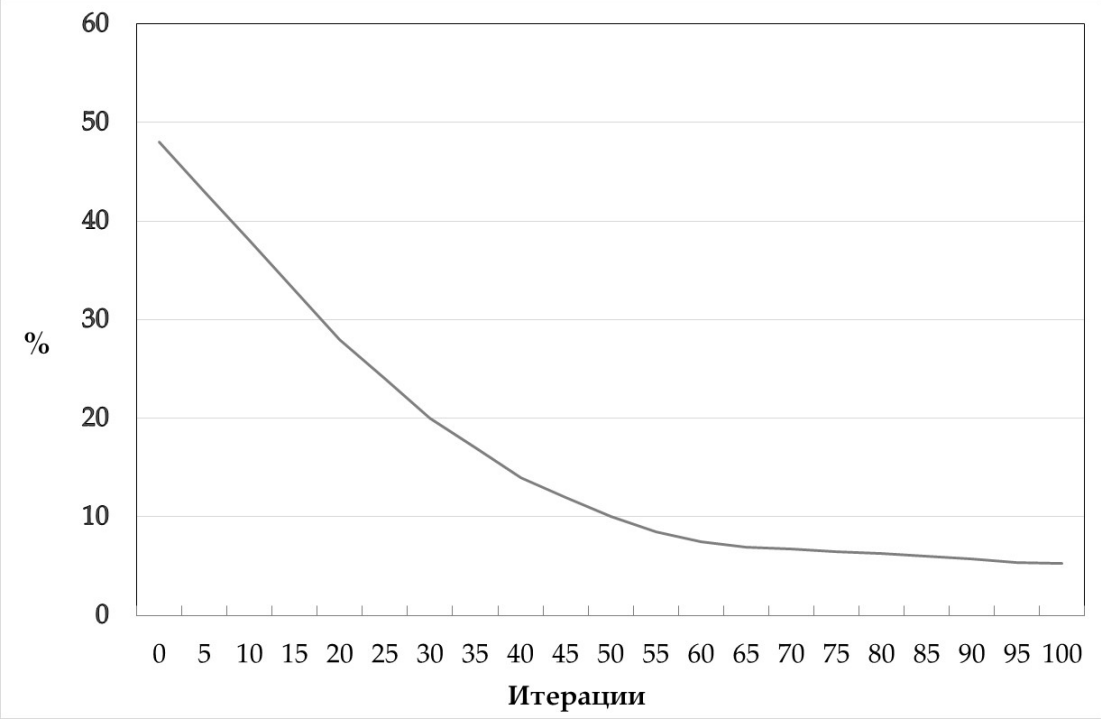


*Рисунок 16. Доля неудачных взаимодействий для первых 100 итераций типичного выполнени последовательной версии модели*

100 Как видно, доля неудачных взаимодействий монотонно уменьшается и стабилизируется примерно на пятипроцентном уровне. Иными словами, механизм обмена мнениями приводит к тому, что мнения агентов становятся более похожими и, таким образом, они чаще попадают в рамки установленного порога и доля успешных взаимодействий увеличивается. Тем не менее, число неудачных взаимодействий никогда не упадет до нуля из-за существования агентов, имеющих сильно отличающиеся численные значения мнений.

101

### 3.1. Параллельная версия модели с взаимодействующими в пространстве агентами для многоядерных архитектур

Разработанная модель обладает следующими особенностями: (1) большое количество пространственно распределенных агентов, (2) высокая децентрализация взаимодействий между агентами, для которой хорошо подходит метод декомпозиции областей. В этой связи при декомпозиции использовался крупнозернистый параллелизм (coarse-grained), предусматривающий достаточно редкий обмен информацией между потоками.

102 При декомпозиции областей ландшафт делится на равные части, каждая из которых обслуживается отдельным потоком, соответствующим одному процессорному ядру. Как уже говорилось, в модели успешное взаимодействие между парой агентов влияет на их мнения. Если успешное взаимодействие произошло между агентами, находящимися в разных областях, то такое взаимодействие будем называть *межпотоковым*, а в противном случае *внутрипотоковым*. За счет обмена данными между потоками и затратой времени на синхронизацию, межпотоковые взаимодействия негативно влияют на производительность. Когда два или более потока одновременно выполняют запись в одном и том же адресном пространстве общей памяти, то это может привести к некорректному выполнению исходной задачи. Для исключения данной ситуации были использованы механизмы взаимного исключения потоков, позволяющие в рамках модели последовательное обновление мнения каждого агента. Вместе с тем, следует отметить, что взаимное исключение в ряде случаев снижало эффективность распараллеливания.

103 **Реализация модели** Параллельная версия агентной модели была реализована в C++ с использованием интерфейса OpenMP, предназначенного для программирования многопоточных приложений на многопроцессорных кластерах с общей памятью. Производительность приложения тестировалась с помощью двух метрик – ускорения и эффективности. Ускорение – это отношение времени выполнения последовательной версии модели к ее параллельной реализации (уравнение 5). В свою очередь эффективность – это отношение ускорения к числу использованных вычислительных узлов (уравнение 6).

104 $$S = \frac{T_s}{T_p} \quad (5)$$

105 $$E = \frac{S}{N} \quad (6)$$

106 где $T_S$ – время выполнения последовательной версии алгоритма, а $T_p$ – время выполнения параллельной версии, использующей $N$ вычислительных узлов.

107

### 3.2. Эксперименты

В процессе оценки эффективности разработанной модели рассматривалось влияние пространственных характеристик отдельных взаимодействий: размер пространства, связанного с количеством пространственно распределенных агентов, участвующих во взаимодействиях (определяет общий объем межагентных взаимодействий) и радиус взаимодействия (определяет максимальное расстояние, на котором пара агентов может связываться). Также в рамках экспериментов тестировалась эффективность масштабирования модели.

108 И, наконец, отметим, что каждая ячейка пространства может быть связана только с одним агентом, а спецификация агентов модели осуществляется с помощью параметров, значения которых приведено в таблице 4. Тысяча итераций была выполнена в рамках одной симуляции для отслеживания динамики системы, а учитывая стохастическую природу агентной модели, для одного эксперимента проводилось 30 симуляций.

109 Таблица 4 Спецификация параметров агентов, обменивающихся мнениями

| **Параметр** | **Значение** |
|---|---|
| Коэффициент $\alpha$ | 0.1 – 1.5 |
| Параметр обучения | 0.2 |
| Порог восприятия мнения | 0.3 |
| Значение мнения | 0.0 – 1.0 |

110 ***Первая серия экспериментов. Оценка влияния размера пространства*** В первой серии экспериментов оценивалось влияние размера пространства на производительность модели и ее масштабируемость. Всего было произведено пять серий расчетов для пространств размерностью от 1000×1000 до 5000×5000 с шагом 1000×1000. Коэффициент α для всех расчетов равен 0.9 и, соответственно, диапазон взаимодействия агентов равен 63 ячейкам (согласно данным таблицы 3).

111 Было установлено, что при увеличении числа ядер ускорение увеличивается, но снижается эффективность. Иными словами, получается, что использование дополнительных аппаратных ресурсов приводит к снижению нагрузки на одно ядро, но при этом, за счет увеличения межпотоковых взаимодействий, эффективность снижается. Что касается размера пространства, то его увеличение снижает показатели эффективности и ускорения для всех расчетов. Разберемся в причинах.

112 Предположим, что все взаимодействия, произошедшие во время одной итерации, оказались успешными. Тогда количество агентов будет хорошим показателем успешных взаимодействий и напрямую связано с общим объемом вычислений. Следуя сделанному предположению, будем использовать показатели пространства 1000×1000 (эталонного пространства) для нормализации результатов, полученных с пространств других размерностей. Отношения

количества взаимодействий других пространств к взаимодействиям на эталонном пространстве показывают относительные объемы вычислений для каждой симуляции (таблица 6). Время расчетов последовательной версии модели (с использованием 1 ядра, таблица 5) с эталонным пространством используется для вычисления отношений времени расчетов модели с пространствами более крупных размеров (таблица 6). Было выявлено, что вычисленные отношения не находятся в абсолютно линейной зависимости. Незначительное расхождение нарастает по мере увеличения размера пространства, что противоречит изначальному ожиданию о линейной зависимости.

113 Таблица 5 Время (в секундах), требующееся для вычислений последовательных и параллельных версий модели с различными размерами пространства и количеством используемых ядер

| Размер пространства | 64 ядра | 128 ядер | 256 ядер | 512 ядер | 1024 ядра |
|---|---|---|---|---|---|
| 1 000×1 000 | 5.30 | 2.85 | 1.55 | 0.86 | 0.48 |
| 2 000×2 000 | 23.68 | 13.51 | 7.93 | 4.82 | 3.02 |
| 3 000×3 000 | 58.27 | 34.85 | 21.70 | 14.19 | 9.72 |
| 4 000×4 000 | 126.86 | 87.22 | 61.67 | 36.83 | 15.96 |
| 5 000×5 000 | 180.25 | 106.35 | 64.72 | 42.78 | 27.35 |

114 Таблица 6 Рассчитанные оценки для последовательных версий модели с различными размерами пространства

| Размер пространства | Количество агентов | Отношение количества взаимодействий к эталонному | Отношение времени вычислений к эталонному |
|---|---|---|---|
| 1 000×1 000 | 1 000 000 | 1 | 1 |
| 2 000×2 000 | 4 000 000 | 4 | 4.07 |
| 3 000×3 000 | 9 000 000 | 9 | 9.16 |
| 4 000×4 000 | 16 000 000 | 16 | 16.52 |
| 5 000×5 000 | 25 000 000 | 25 | 26.38 |

115 Следует отметить, что такое ожидание основано на ранее сделанном предположении об успешности всех взаимодействий. На самом деле это не так, поскольку их успех зависит от используемого в модели порога возможности восприятия стороннего мнения. Таким образом, причина нелинейности заключается в изменении доли успешных взаимодействий.

116 В случае реализации последовательной версии модели, при увеличении размера пространства (и, соответственно, числа агентов) требования к объему памяти резко возрастают, а учитывая фиксированную пропускную способность между процессором и основной памятью, это приводит к увеличению объема трафика. При использовании нескольких ядер для тех же симуляций (с увеличением размера пространства) также наблюдается снижение производительности из-за необходимости обеспечивать согласованность кэш-памяти используемых процессоров.

117 ***Вторая серия экспериментов. Оценка влияния радиуса взаимодействия*** Во второй серии экспериментов оценивалось влияние длины радиуса взаимодействия для агентов модели на производительность и масштабируемость приложения. При расчетах менялись значения коэффициента $\alpha$, влияющего на вероятность выбора соседствующего агента для взаимодействия, от 0.1 до 1.5 с шагом 0.2, а размер пространства для всех симуляций был одинаков (4000 × 4000).

Как и в предыдущем случае, при использовании дополнительных вычислительных ядер ускорение увеличивается, а эффективность снижается, т.е. имеет место потеря производительности на дополнительную единицу используемого ресурса. Также отметим, что при увеличении радиуса взаимодействия, снижается как ускорение, так и эффективность.

118 В таблице 7 приведены результаты расчетов последовательных и параллельных версий модели в зависимости от радиуса взаимодействия агентов и количества используемых ядер. Время вычислений в результате увеличения радиуса также возрастает.

119 Таблица 7 Время (в секундах), требующееся для вычислений последовательных и параллельных версий модели с различным радиусом взаимодействия агентов и количеством используемых ядер

| **Коэффициент α** | **64 ядра** | **128 ядер** | **256 ядер** | **512 ядер** | **1024 ядра** |
|---|---|---|---|---|---|
| 0.1 | 77.72 | 40.98 | 21.82 | 11.77 | 6.42 |
| 0.3 | 83.20 | 45.34 | 25.12 | 14.24 | 8.24 |
| 0.5 | 109.12 | 62.58 | 40.13 | 26.98 | 10.63 |
| 0.7 | 92.34 | 53.09 | 31.48 | 19.32 | 12.41 |
| 0.9 | 103.98 | 63.26 | 34.40 | 19.42 | 19.10 |
| 1.1 | 130.03 | 81.69 | 57.98 | 33.47 | 22.47 |
| 1.3 | 138.83 | 89.98 | 60.56 | 37.24 | 32.83 |
| 1.5 | 125.66 | 82.46 | 52.79 | 45.96 | 36.70 |

120 Таким образом, увеличенный радиус взаимодействия открывает больше возможностей для взаимодействия агентов, поиска единомышленников и, как следствие, приводит к более интенсивному обмену мнениями, а, в конечном счете – требует большего времени для вычислений.

121 Что касается параллельных версий модели, то возрастание количества межпотоковых взаимодействий по мере увеличения числа используемых ядер объясняет снижение эффективности приложения. Это также вполне ожидаемо, поскольку расширяя радиус взаимодействия агентов, мы тем самым увеличиваем вероятность контакта текущего агента с агентами, находящимися в других частях пространства, что приводит к увеличению межпотоковых коммуникаций. Видно, что когда радиус взаимодействия минимальный (0.1, что соответствует 2 клеткам), использование 32 ядер не так сильно снижает эффективность, как при максимальном радиусе (при том же числе ядер).

122

## Заключение

Общий вывод по проведенным расчетам заключается в том, что распараллеливание агентных моделей в любом случае позволяет значительно повысить их производительность, однако способы распараллеливания во многом зависят от специфики конкретной модели (частота общения агентов, их концентрация, сложность внутреннего устройства и т.д.).

123 Разрабатываемое научное направление весьма актуально. Так, ежегодно проводятся совещания рабочей группы стран БРИКС по информационно-коммуникационным технологиям (ИКТ) и высокопроизводительным

вычислениям. В прошлом году оно прошло с 13 по 15 мая 2019 года в технологическом парке Итайпу (Фос-ду-Игуасу, Бразилия) под председательством профессора Аугусто Гадельха (Augusto Gadelha) из Бразильской лаборатории научных вычислений, и в нем приняли участие 22 участника из пяти стран БРИКС. Были сформулированы флагманские проекты в сфере ИКТ и высокопроизводительных вычислений, которые перечислены ниже с указанием страны – ответственной за подготовку расширенного описания с определением критериев оценки будущих проектов по данной теме:

124
- Digital Smart Manufacturing (ответственная страна – Китай).
- HPC application for Life sciences, precision medicine and public health (ответственные страны – ЮАР и Китай).
- Integrated Precision Farming (ответственная страна – Бразилия).
- Large Scale Multi-Agent based Simulation of Virtual Society (ответственная страна – Россия).
- Digital Earth modeling (ответственная страна – Индия).

125 Важным обстоятельством является то, что среди стран БРИКС Россия в ближайшие годы будет возглавлять научное направление, связанно с разработкой мультиагентных систем и искусственных обществ с применением высокопроизводительных вычислений. В этой связи, осуществляемые в рамках Программы Президиума РАН «Механизмы обеспечения отказоустойчивости современных высокопроизводительных и высоконадежных вычислений» наработки будут очень востребованы.

---

**Библиография:**

1. Макаров В.Л., Бахтизин А.Р., Сушко Е.Д. Моделирование социально-экономических процессов с использованием суперкомпьютерных технологий [Текст]: монография // В.Л. Макаров, А.Р. Бахтизин, Е.Д. Сушко. – Вологда: ИСЭРТ РАН, 2016.

2. Borges F., Gutierrez-Milla A., Luque E., Suppi R. Care HPS: A high performance simulation tool for parallel and distributed agent-based modeling // Future Generation Computer Systems, Volume 68, 2017, Pages 59-73.

3. Cordasco G., Scarano V., Spagnuolo C. (2018): Distributed MASON: A scalable distributed multi-agent simulation environment // Simulation Modelling Practice and Theory, Volume 89, 2018, Pages 15-34,

4. Gong Z., Tang W., Bennett D.A., Thill J.C. (2013): Parallel agent-based simulation of individual-level spatial interactions within a multi-core computing environment. // International Journal of Geographical Information Science, 27 (6): 1152-1170.

# Experience in implementing a parallel spatially distributed agent-oriented model using a multi-core architecture

**Albert Bakhtizin**
*Central economic and mathematical Institute of the Russian Academy of Sciences*
*Russian Federation, Moscow*
**Valery Makarov**
*Central Economics and Mathematics Institute of the Russian Academy of Sciences*
*Russian Federation, Moscow*
**Bulat Khabriev**
*"RT-Business Development" LLC*
*Russian Federation, Moscow*

**Abstract**

In the course of research to determine the mechanisms for ensuring the fault tolerance of modern high-performance highly reliable computing as applied to social sciences, we modified and tested the approach to parallelizing resource-intensive agent models. This approach was previously proposed by experts from the University of North Carolina and the University of Iowa (USA). As part of the experiments, the performance of the parallel version of the model was tested depending on two fundamental properties of spatial interactive systems: (1) the size of the space associated with the number of agents distributed over it participating in interactions and (2) the radius of interaction.

**Keywords:** agent-based modeling, parallelization of models, distributed environment, distribution of agents, Care HPS, D-MASON

**Date of publication:** 13.03.2020

**Citation link:**

Bakhtizin A., Makarov V., Khabriev B. Experience in implementing a parallel spatially distributed agent-oriented model using a multi-core architecture // Artificial societies. 2020. Vol. 15. Issue 1. URL: https://artsoc.jes.su/s207751800008235-7-1/ DOI: 10.18254/S207751800008235-7

---