ЭКОНОМИКА И МАТЕМАТИЧЕСКИЕ МЕТОДЫ, 2013, том 49, № 4, с. 18–32

ТЕОРЕТИЧЕСКИЕ И МЕТОДОЛОГИЧЕСКИЕ ПРОБЛЕМЫ

ПРИМЕНЕНИЕ СУПЕРКОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ В ОБЩЕСТВЕННЫХ НАУКАХ*

© 2013 г. В.Л. Макаров, А.Р. Бахтизин

(Москва)

Статья содержит краткий экскурс по вопросам применения суперкомпьютерных технологий в общественных науках, в первую очередь – в части технической реализации крупномасштабных агент-ориентированных моделей (АОМ). Суть данного инструмента в том, что благодаря увеличению мощности компьютеров стало возможным описывать поведение многих отдельных фрагментов сложной системы. В результате мечта многих мыслителей научиться объяснять макроявление на основе поведения его составных частей стала воплощаться в реальность. Например, физики, умеющие описывать поведение элементарных частиц, создали компьютерную имитацию действий большого ансамбля таких частиц и стали изучать его поведение в компьютере, а не в жизни. Таким образом появилось понятие искусственной реальности.

В статье мы рассмотрим опыт зарубежных ученых и практиков в запуске АОМ на суперкомпьютерах, а также на примере АОМ, разработанной в ЦЭМИ РАН, проанализируем этапы и методы эффективного отображения счетного ядра мультиагентной системы на архитектуру современного суперкомпьютера.

Ключевые слова: агент-ориентированные модели, параллельные вычисления, суперкомпьютерные технологии.

Классификация JEL: C63, C88.

ВВЕДЕНИЕ

Компьютерное моделирование – широчайшая, интереснейшая и интенсивно развивающаяся область исследований, востребованная сегодня практически во всех сферах человеческой деятельности. Агент-ориентированный подход к моделированию универсален и удобен для прикладных исследователей и практиков в силу своей наглядности, но при этом предъявляет высокие требования к вычислительным ресурсам. Очевидно, что для прямого моделирования достаточно длительных социальных процессов в масштабах страны и планеты в целом нужны весьма значительные вычислительные мощности.

Суперкомпьютеры позволяют на несколько порядков увеличить число агентов и других количественных характеристик (узлов сети, размеров территории) в моделях, первоначально разработанных для использования в обычных настольных компьютерах. Поэтому суперкомпьютерное моделирование является логичным и крайне желательным шагом для тех упрощенных моделей, которые уже прошли успешную практическую апробацию на обычных компьютерах. Увы, специфика архитектуры современных компьютеров вовсе не гарантирует, что программное обеспечение компьютерной модели немедленно заработает – и на суперкомпьютере. Требуется, как минимум, распараллеливание счетного ядра, а зачастую и его глубокая оптимизация, поскольку в ином случае применение дорогостоящего суперкомпьютерного счета скорее всего себя не оправдает.

* Работа выполнена при финансовой поддержке Российского гуманитарного научного фонда (проекты 11-02-00258 и 12-02-00082).

1. ОПЫТ ЗАРУБЕЖНЫХ УЧЕНЫХ И ПРАКТИКОВ

1.1. Известные примеры запусков агент-ориентированных моделей на суперкомпьютерах. В сентябре 2006 г. стартовал проект по разработке крупномасштабной АОМ европейской экономики – EURACE, т.е. Europe ACE (Agent-based Computational Economics), с очень большим числом автономных агентов, взаимодействующих в рамках социально-экономической системы [Deissenberg, van der Hoog, Herbert, 2008]. В проект вовлечены экономисты и программисты из восьми научно-исследовательских центров Италии, Франции, Германии, Великобритании и Турции, а также консультант из Колумбийского университета США – нобелевский лауреат Джозеф Стиглиц.

Чтобы преодолеть ограничения широко распространенных моделей, рассматривающих агрегированные агенты, а также предполагающих их рациональное поведение и состояние равновесия, в основу исследования была положена методология ACE (Agent Based Computational Economics, агент-ориентированная экономика).

Для модели используется географическая информационная система, охватывающая широкий перечень объектов – предприятия, магазины, школы, транспортные сети и т.д.

Как отмечают разработчики, практически все существующие АОМ рассматривают либо отдельную отрасль, либо относительно небольшой географический район и, соответственно, небольшую популяцию агентов, а в EURACE представлен весь Европейский союз, так что по масштабам и сложности эта модель является уникальной, а ее численное разрешение требует использования суперкомпьютеров, а также специального программного обеспечения.

Для наполнения модели статистической информацией использовались данные (в виде геоинформационных карт) статистической службы Европейского союза уровня NUTS-2[1], представляющие сведения о 268 регионах 27 стран.

В модели – три типа агентов: домашние хозяйства (около 10^7), предприятия (около 10^5) и банки (около 10^2). Все они имеют географическую привязку, а также связаны друг с другом посредством социальных сетей, деловых отношений и т.д.

EURACE реализована с помощью гибкой масштабируемой среды для моделирования агентных моделей – FLAME (Flexible Large-scale Agent Modeling Environment), разработанной Симоном Коакли и Майклом Холкомбом[2] первоначально для моделирования роста клеток, выращиваемых в различных условиях. Используемый в FLAME подход основан на так называемых *X*-машинах, напоминающих конечные автоматы, но отличающихся от них тем, что у каждой *X*-машины есть набор данных (своего рода память), а также тем, что переходы между состояниями являются функциями не только состояния, но и набора данных памяти.

Итак, каждый агент в системе FLAME представлен *X*-машиной, причем предусмотрено общение между агентами посредством передачи сообщений. При работе в параллельном режиме на суперкомпьютере обмен сообщениями между агентами требует больших вычислительных затрат, в связи с чем агенты изначально были распределены по процессорам в соответствии с их географическим положением. Тем самым разработчики программы минимизировали вычислительную нагрузку, исходя из предположения, что в основном общение между агентами происходит в рамках небольшой социальной группы, локализованной в определенной местности. Таким образом, весь модельный ландшафт был поделен на небольшие территории и распределен между узлами суперкомпьютера.

С помощью разработанной модели был проведен ряд экспериментов с целью исследования рынка труда. Не рассматривая детально полученные числовые результаты, отметим, что, по мнению авторов, основной вывод исследования заключается в том, что макропоказатели двух регио-

[1] NUTS (фр. Nomenclature des unités territoriales statistiques) – Номенклатура территориальных единиц для целей статистики, представляющая собой стандарт территориального деления стран для статистических целей, разработанный Европейским союзом и охватывающий входящие в него страны. Существуют NUTS-единицы трех уровней, при этом второй уровень (NUTS-2) соответствует административным округам в Германии, графствам в Великобритании и т.д.

[2] Более подробно см. www.flame.ac.uk.

нов со схожими условиями (ресурсы, развитие экономики и т.д.) в течение продолжительного периода (10 лет и более) могут значительно разойтись за счет первоначальной неоднородности агентов[3].

В АОМ EpiSims, разработанной исследователями из Института биоинформатики Вирджинии (Virginia Bioinformatics Institute), рассматриваются как перемещения агентов, так и их контакты в рамках среды, максимально приближенной к реальности и содержащей дороги, здания и прочие инфраструктурные объекты (Roberts, Simoni, Eubank, 2007). Для построения модели потребовался большой массив данных, включающий информацию о здоровье отдельных людей, их возрасте, доходе, этнической принадлежности и т.д.

Исходная цель исследования заключалась в построении для запуска на суперкомпьютере АОМ большой размерности, с помощью которой можно будет изучать распространение болезней в обществе. Однако впоследствии в процессе работы также решалась задача, связанная с созданием специализированного программного обеспечения ABM++, которое позволяет осуществлять разработку АОМ на языке C++, а также содержит функции, облегчающие распределение исполняемого программного кода на узлах кластеров суперкомпьютера. Помимо этого ABM++ предоставляет возможность динамического перераспределения потоков вычислений, а также синхронизации происходящих событий.

ABM++, первая версия которого появилась в 2009 г., представляет собой результат модернизации инструмента, разработанного в 1990–2005 гг. в Лос-Аламосской национальной лаборатории в процессе построения крупномасштабных АОМ (EpiSims, TRANSIMS, MobiCom).

Межпроцессорные связи между вычислительными узлами в АОМ часто требуют синхронизации происходящих в модели событий. ABM++ позволяет разрабатывать модели, отвечающие этому требованию. К примеру, в социальных моделях агенты часто перемещаются между различными точками пространства (работа, дом и т.д.), а на программном уровне этому соответствует смена узла кластера, и здесь важно, чтобы модельное время принимающего узла было синхронизировано со временем узла, который агент только что покинул.

Также в ABM++ реализована библиотека MPIToolbox, которая соединяет интерфейс C++ API (Application Programming Interface) и Message Passing Interface (MPI) суперкомпьютера и ускоряет передачу данных между узлами кластеров.

ABM++ создавалось в Ubuntu Linux – операционной системе с компиляторами gcc/g++. В качестве интегрированной среды разработки рекомендуется пакет Eclipse с плагином для поддержки C и C++, а также с плагином PTP (Parallel Tools Platform), обеспечивающим разработку и интеграцию приложений для параллельных компьютерных архитектур. Eclipse поддерживает интеграцию с TAU (Tuning and Analysis Utilities) Performance System – инструментом для разностороннего анализа и отладки программ для параллельных вычислений, что также упрощает разработку агентных моделей.

Специалисты другой исследовательской группы из того же Института биоинформатики Вирджинии создали инструмент для изучения особенностей распространения инфекционных заболеваний внутри различных групп населения – EpiFast, к достоинствам которого можно отнести масштабируемость и высокую скорость исполнения. К примеру, имитация социальной активности жителей Большого Лос-Анджелеса (агломерации с населением свыше 17 млн человек) с 900 млн связей между людьми на кластере с 96 двухъядерными процессорами POWER5 заняла менее пяти минут. Такая достаточно высокая производительность обеспечивается предложенным авторами оригинальным механизмом распараллеливания (Bisset, Chen, Feng et al., 2009).

Как правило, для реализации моделирования социума на группе процессоров используют несколько естественных способов. К примеру, один из них базируется на представлении социума в виде набора неструктурированных связей между индивидуумами. При распараллеливании эти связи равномерно распределяются на группы, количество которых соответствует числу процессоров. Минус такого подхода заключается в сложности отслеживания статуса отдельного человека, – иными словами, если, к примеру, инфицирован индивидуум *i*, то информация об этом

[3] Более подробную информацию можно найти на веб-странице проекта: www.eurace.org.

должна быть синхронизирована во всех группах, так как мы не знаем, в каких из них содержатся связи данного человека. Необходимость такой синхронизации влечет за собой большую вычислительную нагрузку. Другой подход основан на разделении людей – агентов модели – в соответствии с их географическим местоположением. Однако в этом случае, поскольку население обычно расселено неравномерно, распределение вычислительной нагрузки требует применения достаточно сложных алгоритмов для ее выравнивания, что также создает дополнительную вычислительную нагрузку.

Предлагаемый авторами метод заключается в равномерном распределении агентов с соответствующими им односторонними исходящими связями по группам, число которых равно числу процессоров. Вычислительный алгоритм основан на взаимодействии ведущих (master) и ведомых (slave) процессоров, организованном следующим образом: ведущий процессор "знает", какой из процессоров обслуживает конкретного агента, а каждый из ведомых процессоров "знает" только обслуживаемых (локальных) агентов. В процессе вычислений ведомые процессоры посылают ведущим процессорам запросы относительно связей с нелокальными агентами. Ведущие процессоры не осуществляют никаких расчетов, кроме поиска нелокального агента для каждой внешней связи, и пересылают запросы соответствующим процессорам.

По мнению разработчиков, предложенный ими алгоритм позволяет проводить эффективные и высокоскоростные симуляции систем с очень большим числом агентов.

Классические модели распространения эпидемий преимущественно основывались на использовании дифференциальных уравнений, однако такой инструментарий затрудняет учет связей между отдельными агентами и их многочисленными индивидуальными особенностями. АОМ позволяют преодолеть указанные недостатки. В 1996 г. Джошуа Эпштейн и Роберт Акстелл опубликовали описание одной из первых АОМ, в которой рассматривается процесс распространения эпидемий (Epstein, Axtell, 1996). Агенты модели, отличающиеся друг от друга восприимчивостью к заболеванию, которая зависит от состояния иммунной системы, распределены на определенной территории. При этом в данной модели агенты, число которых составляет всего несколько тысяч, реализуют достаточно примитивное поведение.

В дальнейшем под руководством Джошуа Эпштейна и Джона Паркера в Центре социальной и экономической динамики Брукингского института (Center on Social and Economic Dynamics at Brookings) была построена одна из самых больших АОМ, включающая данные о всем населении США, т.е. порядка 300 млн агентов (Parker, 2007). Данная модель имеет ряд преимуществ. Во-первых, она позволяет предсказать последствия распространения заболеваний различного типа. Во-вторых, она ориентирована на поддержку двух сред для вычислений: одна среда состоит из кластеров с установленной 64-битной версией Linux, а другая – из серверов с четырехъядерными процессорами и установленной системой Windows (в этой связи в качестве языка программирования был выбран Java, хотя разработчики и не уточняют, какую именно реализацию Java они использовали). В-третьих, модель способна поддерживать численность агентов от нескольких сотен миллионов до 6 млрд.

Способ распределения агентов между аппаратными ресурсами состоит из двух фаз: сначала агенты распределяются по компьютерам, задействованным в работе, а затем по потокам на каждом из компьютеров. В процессе работы модели каждый поток может остановиться (в заранее обусловленное время) для передачи сообщений другим потокам. Все подготовленные к отправке сообщения до определенного момента хранятся в пуле сообщений, а затем одновременно отправляются. Также в реализации модели применяются две вспомогательные утилиты: первая управляет потоками на отдельном компьютере, вторая следит за тем, чтобы все сообщения между потоками были выполнены до момента возобновления вычислительного процесса.

При распределении агентов по аппаратным ресурсам следует учитывать два обстоятельства: 1) производительность узла напрямую зависит от числа инфицированных агентов; 2) контакты, предполагающие передачу сообщений между потоками, требуют гораздо больше вычислительных затрат, чем контакты, ограничивающиеся локальной информацией. Исходя из этого, возможны различные распределения агентов. С одной стороны, можно поделить все рассматриваемое географическое пространство на равные части, число которых должно соответствовать числу узлов, а затем определить для каждого узла какой-либо географический регион. Такое распре-

деление позволяет сбалансировать вычислительную нагрузку между узлами. С другой стороны, можно закрепить определенную территорию, представляющую собой единую административную единицу, за конкретным узлом – в этом случае вычислительную нагрузку удастся сократить за счет снижения числа контактов, предполагающих передачу сообщений между потоками. Если первый способ влечет за собой увеличение вычислительной нагрузки за счет ресурсоемких контактов, то второй в ряде случаев чреват значительным дисбалансом между аппаратными ресурсами. К примеру, вспышка какого-либо заболевания в одном из регионов загрузит один из вычислительных узлов, в то время как некоторые другие будут простаивать.

Рассматриваемая модель (US National Model) включает 300 млн агентов, перемещающихся по карте страны в соответствии с матрицей корреспонденций размерностью 4000×4000, специфицированной с помощью гравитационной модели. На US National Model был проведен вычислительный эксперимент, имитирующий 300-дневный процесс распространения болезни, которая характеризуется 96-часовым периодом инкубации и 48-часовым периодом заражения. В исследовании, в частности, было установлено, что распространение заболевания идет на спад, после того как 65% зараженных выздоровели и приобрели иммунитет. Эту модель неоднократно использовали специалисты Университета Джонса Хопкинса (Johns Hopkins University), а также Департамента национальной безопасности США для исследований, посвященных стратегии быстрого реагирования на различного рода эпидемии (Epstein, 2009).

В 2009 г. была создана вторая версия US National Model, включающая 6,5 млрд агентов, спецификация действий которых проводилась с учетом имеющихся статистических данных. С ее помощью имитировали последствия распространения вируса гриппа A(H1N1/09) в масштабах всей планеты.

Ранее подобная модель была разработана в Лос-Аламосской национальной лаборатории (США), и результаты работы с ней были представлены в широкой печати 10 апреля 2006 г. (Ambrosiano, 2006). Для технической реализации модели был использован один из мощнейших по тем временам суперкомпьютеров с именем Pink, состоящий из 1024 узлов, с двумя процессорами с частотой 2,4 ГГц и 2 ГБ памяти на каждом. С помощью этой крупномасштабной модели, включающей 281 млн агентов, были рассмотрены сценарии распространения различных вирусов, в том числе и H5N1, с учетом тех или иных оперативных вмешательств: вакцинации, закрытия школ и введения карантина на некоторых территориях.

Естественно, для подобных задач можно использовать не только суперкомпьютеры, но и более дешевые решения. Показателен пример ученых Университета Ньюкасла из Австралии (The University of Newcastle, Australia), построивших кластер, включающий 11 узлов – отдельных персональных компьютеров, объединенных в сеть с пропускной способностью до 100 Мбит/с (Lynar, Herbert, Chivers, 2009). Программное обеспечение узлов кластера было одинаковое: операционная система Debian GNU/Linux и JavaParty – кроссплатформенное программное расширение языка Java (ПО JavaParty располагает средствами для распределения потоков выполнения кода по узлам кластеров). Для оценки производительности кластера были использованы три версии одной АОМ, в рамках которой тестировались различные стратегии агентов, принимающих участие в аукционе. Версии различались сложностью стратегий агентов.

Основной вывод тестирования заключается в том, что использование кластеров оправдано лишь тогда, когда вычислительная нагрузка отдельных узлов достаточно интенсивная; в противном случае распределение потоков, наоборот, уменьшает скорость работы модели. Отметим, что такое поведение кластерных приложений неспецифично.

1.2. RepastHPC – среда проектирования агент-ориентированных моделей для высокопроизводительных вычислений. Отдельного внимания заслуживает программное обеспечение, разработанное для проектирования АОМ с целью последующего запуска на суперкомпьютерах, – Repast for High Performance Computing (RepastHPC). Данный пакет реализован с использованием языка C++ и MPI – программного интерфейса для обмена сообщениями между процессами, выполняющими задачу в параллельном режиме, а также библиотеки Boost, расширяющей C++.

В рамках RepastHPC реализован динамический дискретно-событийный планировщик выполнения программных инструкций с консервативными алгоритмами синхронизации, предусматривающими задержку процессов для соблюдения определенной очередности их выполнения.

В RepastHPC агенты распределяются между процессами, и каждый процесс связан с агентом, являющимся *локальным* по отношению к данному процессу. В свою очередь, агент локален к процессу, выполняющему программный код, описывающий поведение данного агента. При этом копии остальных – *нелокальных* – агентов могут присутствовать в любом процессе, что позволяет агентам всей модели взаимодействовать с этими копиями. К примеру, пусть пользователь в своей модели, предполагающей параллельные вычисления, использует два процесса – P1 и P2, каждый из которых создает определенное количество агентов и имеет собственный планировщик выполнения программных инструкций. Агенты, поведение которых рассчитывается на процессе P1, являются локальными по отношению к данному процессу, и только в рамках данного процесса программный код может изменить их состояние (аналогично и для процесса P2). Предположим, процесс P1 запрашивает копию агента A2 из процесса P2. Агент A2 не является локальным по отношению к процессу P1, и, соответственно, программный код, выполняемый в рамках процесса P1, не может изменить состояние агента A2. При этом агенты, реализуемые в рамках процесса P1, при необходимости могут запросить состояние агента A2, но копия A2 останется неизменной. Изменение оригинального A2 возможно только в рамках процесса P2, но в этом случае RepastHPC синхронизирует изменения состояния агента между всеми процессами[4].

2. АДАПТАЦИЯ АГЕНТ-ОРИЕНТИРОВАННЫХ МОДЕЛЕЙ ДЛЯ СУПЕРКОМПЬЮТЕРА: НАШ ПОДХОД

В марте 2011 г. на суперкомпьютере “Ломоносов” была запущена модель, имитирующая развитие социально-экономической системы России на протяжении последующих 50 лет. Эта АОМ основана на взаимодействии 100 млн агентов, условно представляющих социально-экономическую среду России. Поведение каждого агента задано набором алгоритмов, которые описывают его действия и взаимодействие с другими агентами в реальном мире.

В проекте участвовали пять человек: два специалиста ЦЭМИ РАН (В.Л. Макаров, А.Р. Бахтизин) и три сотрудника МГУ (В.А. Васенин, В.А. Роганов, И.А. Трифонов). Данные для моделирования были предоставлены Федеральной службой государственной статистики и Российским мониторингом экономического положения и здоровья населения. Модель для обычного компьютера была построена в 2009 г., а в 2011 г. она была конвертирована в суперкомпьютерную версию. Ниже мы рассмотрим основные этапы и методы эффективного отображения счетного ядра мультиагентной системы на архитектуру современного суперкомпьютера, наработанные нами в процессе решения соответствующей задачи.

2.1. Проблема масштабирования. Важно понимать, что масштабирование программ для суперкомпьютеров является фундаментальной проблемой. Хотя обычная и суперкомпьютерная программы реализуют один и тот же функционал, целевые функции их разработки, как правило, разные.

При начальной разработке сложного прикладного программного обеспечения в первую очередь стараются сократить издержки на программирование, обучение персонала, повысить переносимость между платформами и т.д., а оптимизацию откладывают “на потом”. И это вполне разумно, так как на ранних стадиях приоритетом разработки является исключительно функционал.

Однако когда разработанное программное обеспечение уже начало внедряться, часто выясняется, что на больших реальных данных не хватает производительности. А поскольку современные суперкомпьютеры – это вовсе не разогнанные в тысячи раз персональные компьютеры, для запуска на суперкомпьютере программу приходится существенно видоизменять. Причем эффективно сделать это без специальных знаний и навыков удается далеко не всегда.

[4] Более подробно с данным ПО можно ознакомиться в руководстве пользователя (Collier, 2012), а по адресу http://repast.sourceforge.net/repast_hpc.html – скачать версию пакета 1.0.1 (по состоянию на 5 марта 2012 г.).

При грамотно проведенной работе значительное повышение эффективности обычно достигается на трех уровнях:

1) распараллеливание счета;

2) специализация вычислительных библиотек по задачам;

3) низкоуровневая оптимизация.

2.2. Специализация и низкоуровневая оптимизация. Прежде чем всерьез говорить об использовании суперкомпьютеров, программу следует максимально оптимизировать и адаптировать к целевой аппаратной платформе. Если этого не сделать, параллельная версия будет лишь хорошим тестом для суперкомпьютера, а сам счет будет весьма неэффективным.

Использовать суперкомпьютер без оптимизации и адаптации программы к целевой аппаратной платформе – все равно что послать на боевое задание полк новобранцев: их нужно сначала хорошо обучить выполнять их задачу (специализация, оптимизация программного обеспечения), а также эффективно владеть оружием (низкоуровневая оптимизация программного обеспечения), и вот тогда это будет эффективным использованием ресурсов.

В универсальных системах моделирования типа AnyLogic предоставляемые процедуры универсальны. А универсальный код часто можно оптимизировать для конкретного семейства задач.

2.3. Выбор системы поддержки моделирования. Безусловно, АОМ можно программировать и без специальной среды, на любом объектно-ориентированном языке. При этом основным недостатком существующих пакетов для создания АОМ, кроме RepastHPC, является невозможность разработки проектов, выполняющихся на вычислительном кластере (т.е. не предусмотрен механизм распараллеливания процесса выполнения программного кода).

Однако более разумным подходом будет использование одной из хорошо зарекомендовавших себя систем для АОМ – по причине унифицированной реализации типичных способов взаимодействия агентов. В целях экономии места здесь мы рассмотрим только систему ADEVS[5].

ADEVS представляет собой набор низкоуровневых библиотек для дискретного моделирования, выполненных на языке C++. Из достоинств стоит отметить:

– простоту реализации;

– высокую производительность моделей;

– поддержку основных численных методов при построении моделей;

– встроенное распараллеливание симуляции при помощи OpenMP;

– возможность применения стандартных средств распараллеливания;

– достаточно быстрое развитие библиотек в текущий момент;

– кроссплатформенность;

– низкоуровневость (текущий функционал не накладывает никаких ограничений на модель);

– независимость скомпилированного кода от нестандартных библиотек;

– открытый исходный код.

Однако существенными недостатками этого продукта являются полное отсутствие средств презентации и достаточно сложная по сравнению, например, с AnyLogic разработка моделей. Поэтому этот продукт не может использоваться для построения моделей на уровне заказчика, однако представляет собой эффективную платформу для реализации параллельных симуляторов.

Основными элементами программы с использованием библиотеки ADEVS при построении АОМ обычно являются:

– симулятор adevs:: Simulator<X>;

– примитив агентов adevs:: Atomic<X>;

– модель (контейнер агентов) adevs:: Digraph<VALUE, PORT>.

[5] Систему ADEVS и ее описание можно скачать отсюда: http://www.ornl.gov/~1qn/adevs/.

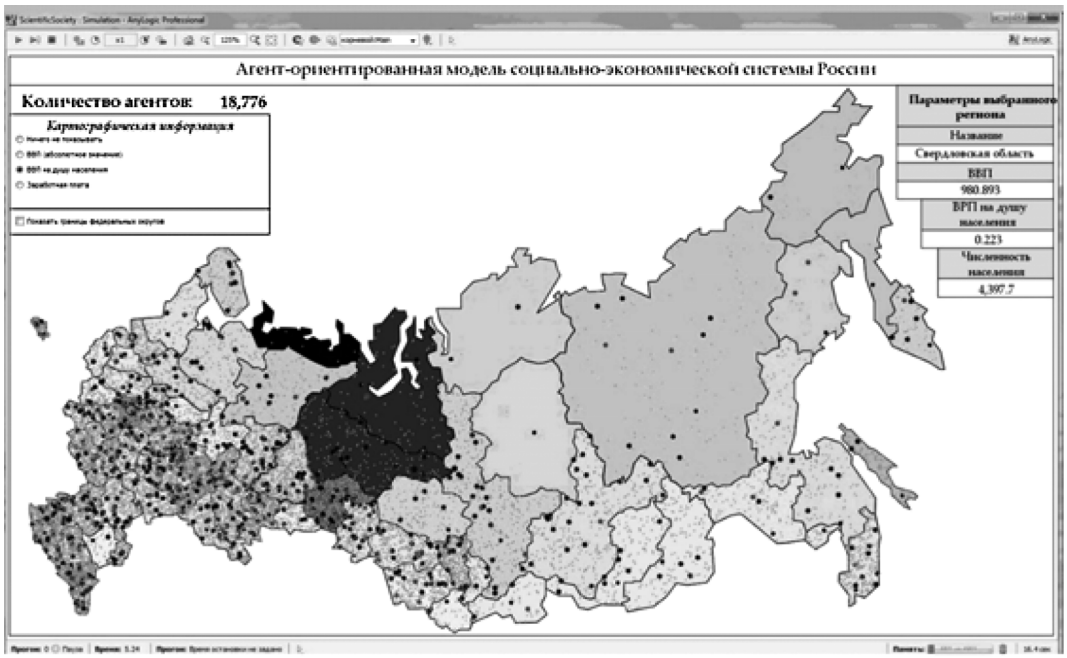

Рис. 1. Рабочее окно АОМ

В силу перечисленных выше достоинств системы ADEVS суперкомпьютерную версию описываемой ниже программы было решено реализовывать именно на ее базе. В рамках данной работы были разработаны MPI-версия симулятора ADEVS, а также система визуализации процесса счета на базе библиотеки Qt – кросс-платформенного инструментария разработки ПО на языке программирования C++.

Далее мы переходим к краткому описанию разработанной модели и процедуры ее последующего запуска на суперкомпьютере.

2.4. Исходная агент-ориентированная модель. Первый этап разработки описываемой ниже АОМ заключался в построении инструмента, эффективно решающего задачу исследования на обычных компьютерах, а также в настройке параметров модели. После ее успешной апробации с небольшим числом агентов (около 20 тыс. – такова численность агентов, над которой способен производить вычисления с удовлетворительной скоростью персональный компьютер с хорошей производительностью, учитывая сложность агентов) было решено конвертировать модель для суперкомпьютера – в этом состоял второй этап разработки. На первом этапе был использован пакет AnyLogic, технические возможности которого позволили достаточно быстро отладить модель и настроить ее параметры. Таким образом, модель для обычного компьютера была построена в 2009 г., а в 2011 г. она была конвертирована в суперкомпьютерную версию.

На рис. 1 изображено рабочее окно разработанной АОМ (точки – агенты). В процессе работы системы можно получать оперативную информацию о социально-экономическом положении всех регионов России, в том числе с использованием картографической информации, меняющейся в режиме реального времени в зависимости от значений эндогенных параметров.

Спецификация агентов модели осуществлялась с учетом следующих основных параметров: возраст; продолжительность жизни; специализация родителей; место работы; регион проживания; доход и др.

Спецификация регионов осуществлялась с учетом следующих параметров: географические границы; число жителей; число работников (по типам); ВРП; ВРП на душу населения; объем инвестиций; объем инвестиций на душу населения; средняя заработная плата; средняя продолжительность жизни; показатель прироста населения и др.

Для наполнения модели данными использовались статистические сборники Росстата, а также социологические базы данных RLMS.

2.5. Конвертация модели в суперкомпьютерную программу. Выше мы уже обсуждали проблемы, связанные с использованием средств разработки АОМ для реализации проектов, выполняющихся на вычислительных кластерах суперкомпьютера. У AnyLogic ввиду трудностей отделения счетной части от презентативной и из-за реализации кода на языке высокого уровня Java производительность выполнения кода существенно ниже, чем у ADEVS, и, кроме того, очень сложно (либо очень трудоемко) переработать генерируемый код в параллельно выполняемую программу.

Ниже приведен алгоритм конвертации модели AnyLogic в суперкомпьютерную программу.

2.6. Трансляция модели. Модели в проекте AnyLogic хранятся в виде XML-файла, содержащего дерево необходимых для генерации кода параметров: классы агентов, параметры, элементы презентации, описания UML-диаграмм поведения агентов.

В процессе работы конвертера это дерево транслируется в код C++ программы, вычисляющей эту модель. Проход дерева совершается “в глубину”, при этом выделяются следующие ключевые стадии, и происходит их совмещение с выполнением задачи трансляции.

1. *Генерация основных параметров.* Поиск корня дерева и считывание параметров дочерних вершин, таких как имя модели, адрес сборки, тип модели, тип презентации.

2. *Генерация классов.* Генерация классов (более подробно):

1) построение списка классов;

2) считывание основных параметров класса;

3) считывание переменных;

4) считывание параметров;

5) считывание функций;

6) генерация списка функций;

7) считывание кода функций;

8) преобразование кода функций Java –> C++;

9) считывание используемых фигур и элементов управления;

10) генерация кода инициализации фигур и элементов управления;

11) генерация кода конструктора, деструктора, визуализатора;

12) генерация структуры класса;

13) генерация кода заголовочного и source-файлов.

3. *Генерация симулятора.* Поиск вершины, хранящей информацию о процессе симуляции (управляющие элементы, значения важных констант, элементы презентации и т.д.).

4. *Генерация общих файлов проекта* (main.cpp, mainwindow.h, mainwindow.cpp и т.д.).

2.7. Импортирование входных данных. В качестве выходных данных в модель загружаются данные из геоинформационной составляющей исходной модели (карты России), содержащей всю необходимую информацию.

2.8. Генерация классов и преобразование кода функций. При генерации функций из дерева считываются: имя функции, возвращаемый тип, параметры и ее тело.

На основании построенного ранее списка классов в аргументах функций производится замена “тяжеловесных классов”, т.е. всех сгенерированных классов, классов фигур и прочих классов, не входящих в стандартное множество, на соответствующие указатели с целью экономии памяти и во избежание ошибок при работе с нею. Далее генерируются заголовки функций, которые впоследствии вставляются в заголовочные и source-файлы. При таком считывании тело функции посредством соответствующей функции (листинг 5) преобразуется из Java-кода в аналогичный ей C++ код (это возможно ввиду достаточно узкого класса используемых функций, а в случае более сложных функций требуется ручная доработка транслированного кода), после чего она добавляется в список тел для данного класса.

В ходе трансляции неоднократно возникает задача преобразования исходного кода функций из языка Java в язык C++. Его можно представить в виде последовательных замен конструкций, например таких.

– *Преобразование циклов:* Java-формат.

– – *Преобразование указателей.* В Java нет столь явного, как в C++, различия между объектом и указателем на объект, поэтому структуры работы с ними не отличаются. Поэтому вводится список классов, в которых важно использовать операции с указателем на объект, а не с самим объектом, и отслеживаются все переменные этих классов с последующей заменой в пределах данной функции при обращении к ним на соответствующие обращения к указателям.

– – *Раскрытие "черных ящиков".* В Java, и конкретно в библиотеке AnyLogic, есть некоторое число функций и классов, аналогов которым нет ни в самом C++, ни в библиотеке ADEVS. В связи с этим были созданы дополнительные библиотеки shapes.h, mdb-work.h, в которых и реализованы недостающие функции.

– – *На этапе генерации основных параметров списка классов* получаются название основного класса и названия моделируемых классов-агентов. В код основного класса добавляется процедура добавления агента в зону видимости симулятора.

2.9. Генерация внешних объектов. В процессе генерации внешних объектов создается отдельная функция Main::initShapes(), в которой содержится вся "графическая информация", т.е. в рамках функции происходит инициализация всех фигур, классы которых реализованы также в shapes.h. Соответствующий пример приведен в следующем отрывке кода.

2.10. Генерация классов, кода заголовочного и source-файлов. На основе всех считанных и сгенерированных данных создаются заголовочный и source-файл соответствующего класса.

2.11. Генерация симуляции. Для генерации симуляции оказалось достаточно заранее написанных файлов main.cpp, mainwindow.cpp, mainwindow.h, в которых шаблоны задают тип основного класса и подключаемые заголовочные файлы. При компоновке исходного кода шаблоны заменяются на полученные ранее (на стадии генерации) названия классов. Этого достаточно для двухпоточной симуляции, которую позднее можно заменить на соответствующий модуль для многопроцессорной симуляции.

2.12. Дополнительные параметры. На этапе разбора дерева (см. выше) параллельно формируется сходное по строению дерево для генерации C++ кода, с помощью которого на этапе подготовки к трансляции можно задать необходимые параметры компиляции (визуализацию тех или иных частей, визуальную валидацию распознавания кода, дополнительные флаги сборки и т.д.).

После этого при команде трансформации с учетом данных параметров и происходит окончательная компиляция.

2.13. Сборка готового проекта. Для сборки транслированного проекта применяется QtCreator – кроссплатформенная свободно распространяемая интегрированная среда разработки для работы с фреймворком Qt.

2.14. Код агента. При помощи описанного выше транслятора из данных файлов проекта AnyLogic (model.alp и др.) был сгенерирован исходный код (за исключением характера поведения агента).

Характер поведения агента должен быть сгенерирован из диаграммы состояний, однако на данный момент автоматизация этого процесса еще не реализована. Таким образом, к сгенерированному коду необходимо было добавить еще некоторый объем кода.

После внесения необходимых изменений в результате компиляции получилось кроссплатформенное приложение, повторяющее основную функциональность данной модели.

2.15. Статистика и визуализация временных срезов. Ввиду неинтерактивности запуска программы на больших суперкомпьютерах сбор выходных данных и визуализация были разделены (это связано с неравномерностью нагрузки на кластеры в разное время суток, а монопольный

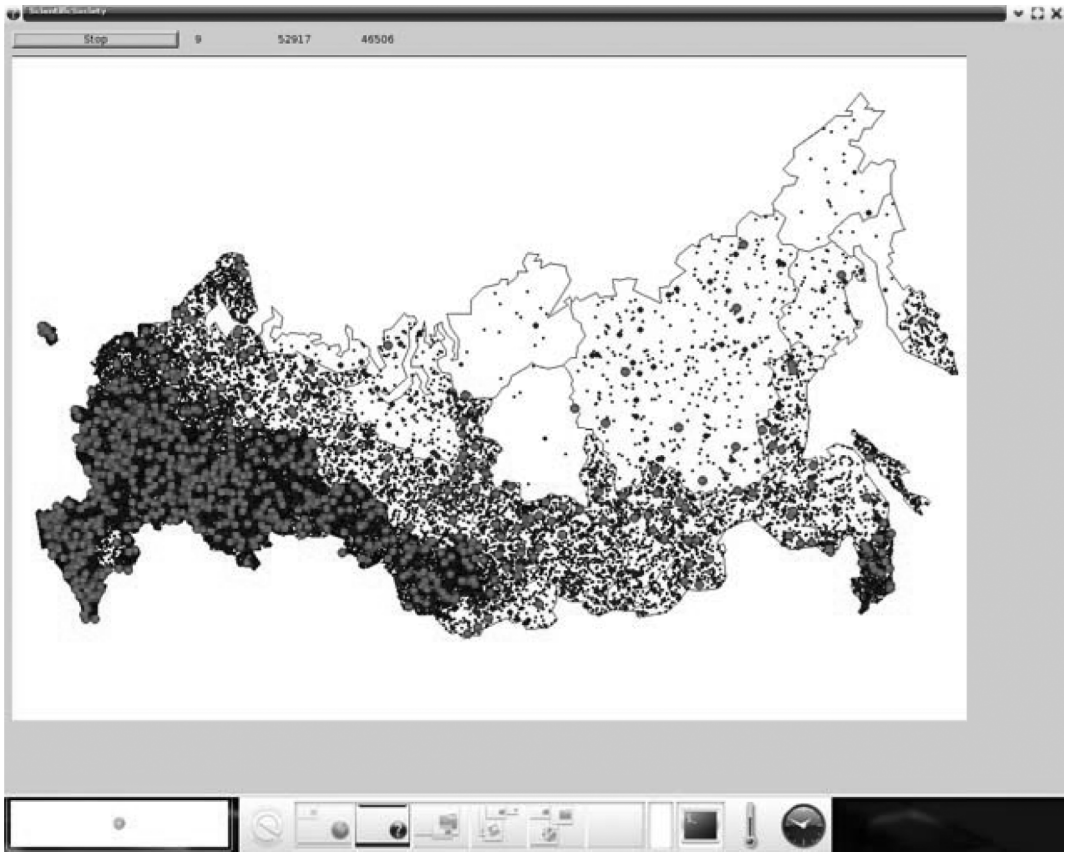

Рис. 2. Результат работы суперкомпьютерной программы в графическом виде

доступ попросту невозможен). После пересчета модели получившаяся на выходе информация может быть снова визуализирована, к примеру, следующим образом (рис. 2).

2.16. Доступные для расчетов суперкомпьютеры. На момент проведения расчетов для нас были доступны три суперкомпьютера (таблица), входящие в первую пятерку суперкомпьютерного рейтинга Топ-50 в странах СНГ (в редакции от 29.03.2011).

Мы использовали для расчетов два суперкомпьютера – “Ломоносов” и МВС-100К.

2.17. Результаты. За счет применения суперкомпьютерных технологий и оптимизации программного кода удалось достичь очень высокой производительности.

Сама по себе оптимизация кода и использование C++ вместо Java позволили увеличить скорость выполнения программы. Так, модель тестировалась при следующих начальных условиях: 1) число агентов – 20 тыс.; 2) прогнозируемый период – 50 лет. По итогам расчетов оказалось, что время счета модели с использованием ADEVS составило 48 с на одном процессоре, а время счета модели также на одном процессоре с использованием AnyLogic составило 2 мин 32 с, – а это значит, что среда разработки была выбрана удачно.

Таблица. Доступные для исследовательской группы суперкомпьютеры

Позиция в Топ-50	Суперкомпьютеры	Узлы	CPU	Ядра	RAM/узел	TFlops
1	“Ломоносов” (МГУ)	5130	10 260	44 000	12GB	414
4	МВС-100К (МСЦ РАН)	1256	2332	10 344	8GB	123
5	“Чебышев” (МГУ)	633	1250	5000	8GB	60

Как отмечалось выше, обычный персональный компьютер с хорошей производительностью способен осуществлять вычисления с удовлетворительной скоростью над совокупностью агентов числом около 20 тыс. (поведение каждого из них задается приблизительно 20 функциями), и при этом среднее время пересчета одной единицы модельного времени (один год) составляет около минуты. При большем числе агентов, к примеру 100 тыс., компьютер попросту "зависает".

Задействование 1000 процессоров суперкомпьютера и выполнение оптимизированного кода позволило увеличить число агентов до 100 млн, а число модельных лет – до 50. При этом такой гигантский массив вычислений был выполнен за период времени, приблизительно равный 1 мин 30 с (этот показатель может несколько меняться в зависимости от типа используемых процессоров).

Результаты моделирования показали, что при сохранении текущих тенденций население Сибирского и Дальневосточного федеральных округов значительно сократится, в то время как в регионах Южного федерального округа, наоборот, произойдет существенный прирост. Кроме того, проведенное моделирование дает основания прогнозировать постепенное снижение ВВП, а также ряда других макроэкономических показателей.

По результатам экспериментов с разработанной АОМ выяснилось, что масштабирование модели само по себе имеет определенное значение. Так, при запуске одной и той же версии модели на 50 модельных лет с одинаковыми параметрами (за исключением числа агентов: в первом случае было 100 млн агентов, а во втором – 100 тыс. агентов) были получены результаты, а именно масштабированное число агентов, отличающиеся приблизительно на 4,5%.

Можно предположить, что в сложных динамических системах одни и те же параметры (рождаемость, продолжительность жизни и т.д.) могут приводить к разным результатам в зависимости от размера социума.

ВЫВОДЫ

Хотя подобная работа может быть проделана практически для любой начальной реализации, в идеале нужны системы, которые ликвидировали бы этот непростой этап, который сегодня затрудняет широкое внедрение суперкомпьютерного моделирования.

Тот факт, что в исходном представлении модели значительную роль играет код на языке программирования, отражает фундаментальное свойство агент-ориентированного подхода: модель – это программа.

Декларативный подход (набор правил вместо обычной программы) может сильно упростить задачу определения поведения системы.

Проиллюстрируем вышесказанное (рис. 3). При возникновении новой задачи мы получаем для нее некоторые исходные данные, а аналитики определяют, к какому классу задач она относится, с целью подбора соответствующей спецификации для построения соответствующей модели. Для этого используется наиболее адекватный DSL (Domain Specific Language), который позволяет описывать модели (в качестве IDE, к примеру, может быть применен Eclipse). Для этого языка, как правило, уже имеется ряд наработок на прошлых задачах этого класса, которые содержат уже адаптированные специфические библиотеки. В совокупности все это образует метаязык, который затем реализуется в некоторой среде разработки, и именно в ней, с точки зрения прикладников, в итоге и решается задача. После тестовых прогонов готовой программы на локальной машине следует финальная стадия – запуск на суперкомпьютере (Макаров, Бахтизин, Васенин и др., 2011).

При этом общая последовательность действий разработки будет следующей (рис. 4).

1. Выбор языка описания модели (т.е. метамодели и языка описания моделей). Первый шаг контролируется специалистами формализации моделей и собственно программистами, которые выбирают язык, наиболее подходящий для описания данной задачи.

2. Описание модели на выбранном языке (этот шаг контролируется заказчиком).

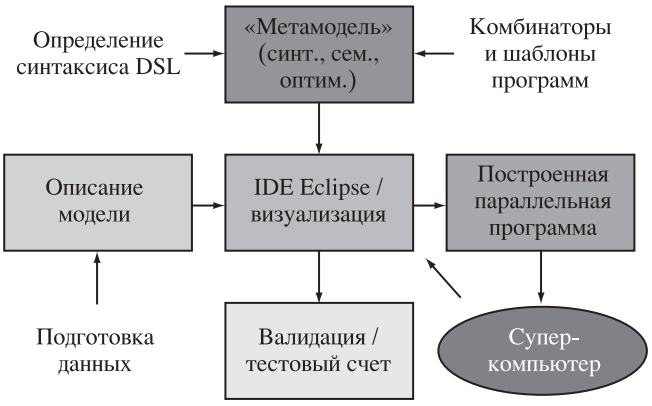

Рис. 3. Процесс разработки агентной модели для суперкомпьютера

3. Поиск и выбор необходимых шаблонов программирования, т.е., по сути, выбор удобной среды разработки с наиболее подходящим инструментарием.

4. Подготовка алгоритмов распараллеливания и выбор аппаратной части.

5. Выбор библиотек, прочей программной части и непосредственно программирование.

На четвертом и пятом шагах происходит перевод модели в низкоуровневые языки, подходящие для распараллеливания, и осуществляется собственно запуск программы, настройка средств отображения результатов и представление их заказчику (после чего возможны дополнительные итерации).

Итак, мы рассмотрели наиболее известный и успешный опыт зарубежных ученых и практиков запуска АОМ на суперкомпьютерах, а также описали этапы и методы эффективного отображения счетного ядра мультиагентной системы на архитектуру современного суперкомпьютера на примере агентной модели, разработанной в ЦЭМИ РАН.

Как было показано, суперкомпьютеры позволяют на несколько порядков увеличить число агентов и других количественных характеристик (количество узлов сети, размер территории) в моделях, первоначально разработанных для использования на обычных настольных компьютерах.

Можно сделать однозначный вывод о том, что суперкомпьютерное моделирование является логичным и крайне желательным шагом для тех упрощенных моделей, которые уже прошли успешную практическую апробацию на обычных компьютерах.

Вместе с тем прямое перенесение компьютерной модели на суперкомпьютер практически невозможно, так как значительное повышение эффективности обычно достигается на трех уровнях:

1) распараллеливание счета;

2) специализация вычислительных библиотек по задаче;

3) низкоуровневая оптимизация.

При этом возникает принципиальная проблема, связанная с интенсивным межагентным взаимодействием и заключающаяся в эффективном распараллеливании программного кода между процессорами.

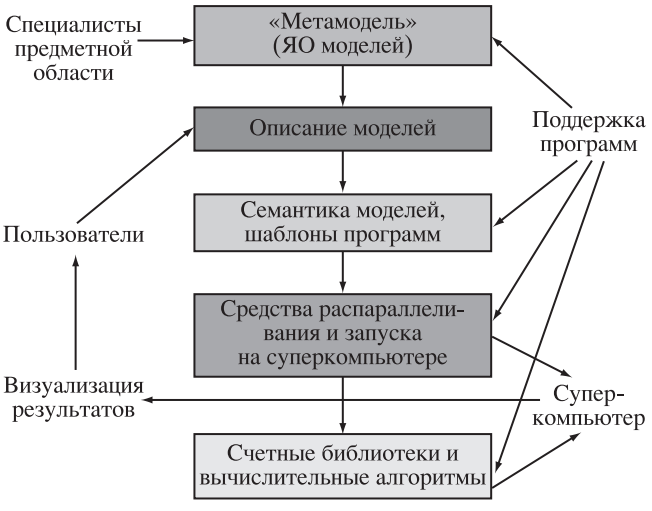

Рис. 4. Структура хранилища, роли и доступ к общему репозиторию

Опыт зарубежных исследователей, равно как и наш, показывает, что существуют два способа ее решения.

Способ 1. При распараллеливании кода агенты модели равномерно распределяются на группы, количество которых соответствует числу процессоров. Минус такого подхода заключается в сложности отслеживания статуса отдельного человека, т.е. если индивидуум i, к примеру, инфицирован (в случае агентной модели распространения эпидемий), то информация об этом должна быть синхронизирована во всех группах, так как мы не знаем, в каких из них содержатся связи данного человека. Необходимость синхронизировать информацию влечет за собой большую вычислительную нагрузку.

Способ 2. Основан на разделении людей – агентов модели – в соответствии с их географическим местоположением (исходя из первоначального предположения о том, что в основном общение между агентами происходит в рамках небольшой социальной группы, проживающей приблизительно в одной местности). В этом случае географическое пространство делится на равные части, число которых соответствует числу узлов.

Если первый способ влечет за собой увеличение вычислительной нагрузки за счет ресурсоемких контактов, то второй чреват опасностью значительного дисбаланса между аппаратными ресурсами. К примеру, вспышка какого-либо заболевания в одном из регионов загрузит один из вычислительных узлов, в то время как другие будут простаивать. Таким образом, конкретный способ распараллеливания зависит от специфики решаемой задачи.

СПИСОК ЛИТЕРАТУРЫ

Макаров В.Л., Бахтизин А.Р., Васенин В.А., Роганов В.А., Трифонов И.А. (2011). Средства суперкомпьютерных систем для работы с агент-ориентированными моделями // Программная инженерия. № 3.

Ambrosiano N. (2006). Avian Flu Modeled on Supercomputer // *Los Alamos National Laboratory NewsLetter*. Vol. 7. No. 8.

Bisset K., Chen J., Feng X., Kumar VSA, Marathe M. (2009). EpiFast: A fast algorithm for large scale realistic epidemic simulations on distributed memory systems. Yorktown Heights, New York; 2009:430–439. Proceedings of 23rd ACM International Conference on Supercomputing (ICS'09).

Collier N. (2012). Repast HPC Manual. [Электронный ресурс] February 23. Режим доступа: http://repast.sourceforge.net, свободный. Загл. с экрана. Яз. англ. (дата обращения: май 2013 г.).

Deissenberg C., Hoog S. van der, Herbert D. (2008). EURACE: A Massively Parallel Agent-Based Model of the European Economy // *Document de Travail No. 2008*. Vol. 39. 24 June.

Roberts D.J., Simoni D.A., Eubank S. (2007). A National Scale Microsimulation of Disease Outbreaks. RTI International. Research Triangle Park. Blacksburg: Virginia Bioinformatics Institute.

Epstein J.M., Axtell R.L. (1996). Growing Artificial Societies: Social Science from the Bottom Up. Ch. V. Cambridge, Massachusetts: MIT Press.

Epstein J.M. (2009). Modeling to Contain Pandemics // *Nature.* Vol. 460. 6 August.

Keith R.B., Jiangzhuo C., Xizhou F., Kumar A.V.S., Madhav V.M. (2009). EpiFast: A Fast Algorithm for Large Scale Realistic Epidemic Simulations on Distributed Memory Systems ICS'09. June 8–12. N.Y.: Yorktown Heights.

Parker J. (2007). A Flexible, Large-Scale, Distributed Agent Based Epidemic Model. Center on Social and Economic Dynamics. Working Paper No. 52.

Lynar T.M., Herbert R.D., Chivers W.J. (2009): Implementing an Agent Based Auction Model on a Cluster of Reused Workstations // *International J. of Computer Applications in Technology*. Vol. 34. Issue 4.

Поступила в редакцию
05.06.2013 г.

Use of the Supercomputer Technologies in Social Sciences

V.L. Makarov, A.R. Bakhtizin

Gives a short digression of the application of supercomputer technologies in social studies, primarily regarding technical realization of the large-scale agent-focused models (AFM). The instrument allows to describe behaviour of many fragments in a complex system thanks to the immense increase in the capacity of computers. As a result a dream of many researchers to learn to explain the macro-phenomenon on the basis of behaviour of its components has began to be embodied in a reality. For example, physicists, able to describe behaviour of elementary particles, have created computer imitation of activities of a big ensemble of such particles and began to study its behaviour using the computer, instead of in life. Thus appeared a concept of an artificial reality.

In the article we consider the experience of foreign scientists and experts in starting AFM on supercomputers, and also the example AFM, developed in CEMI (RAS), and analyze the stages and methods of effective display of a calculating core of the multi-agent system on the architecture of a modern supercomputer.

Keywords: agent-focused model, parallel calculations, supercomputer technologies.

JEL Classification: C63, C88.