ISSN 2220-3397

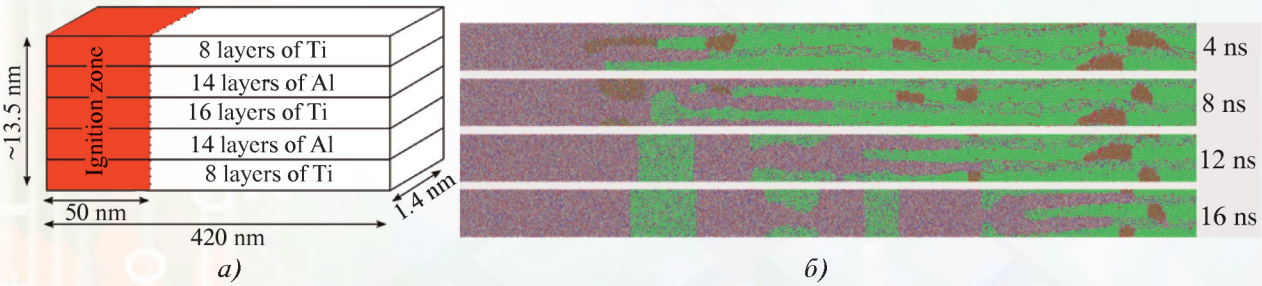
ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Программная инженерия

ПрИн
10
2022
Том 13

Рисунки к статье **В. И. Иордана, И. А. Шмакова**

«3D-ВИЗУАЛИЗАЦИЯ ГЕТЕРОФАЗНЫХ ИНТЕРМЕТАЛЛИДНЫХ СТРУКТУР НА ОСНОВЕ КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ СВС В НАНОРАЗМЕРНОЙ СЛОИСТО-БЛОЧНОЙ СТРУКТУРЕ Ti-Al»



а) *б)*

Рис. 2. Структура слоистой атомной системы Ti-Al (*а*) и временные диаграммы распределения интерметаллических фаз (*б*) [11]

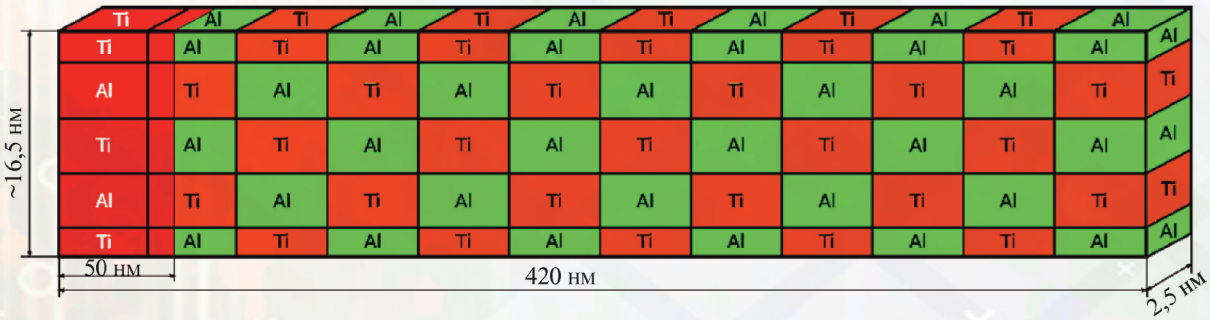


Рис. 3. Структурная схема наноразмерной атомной слоисто-блочной системы Ti-Al

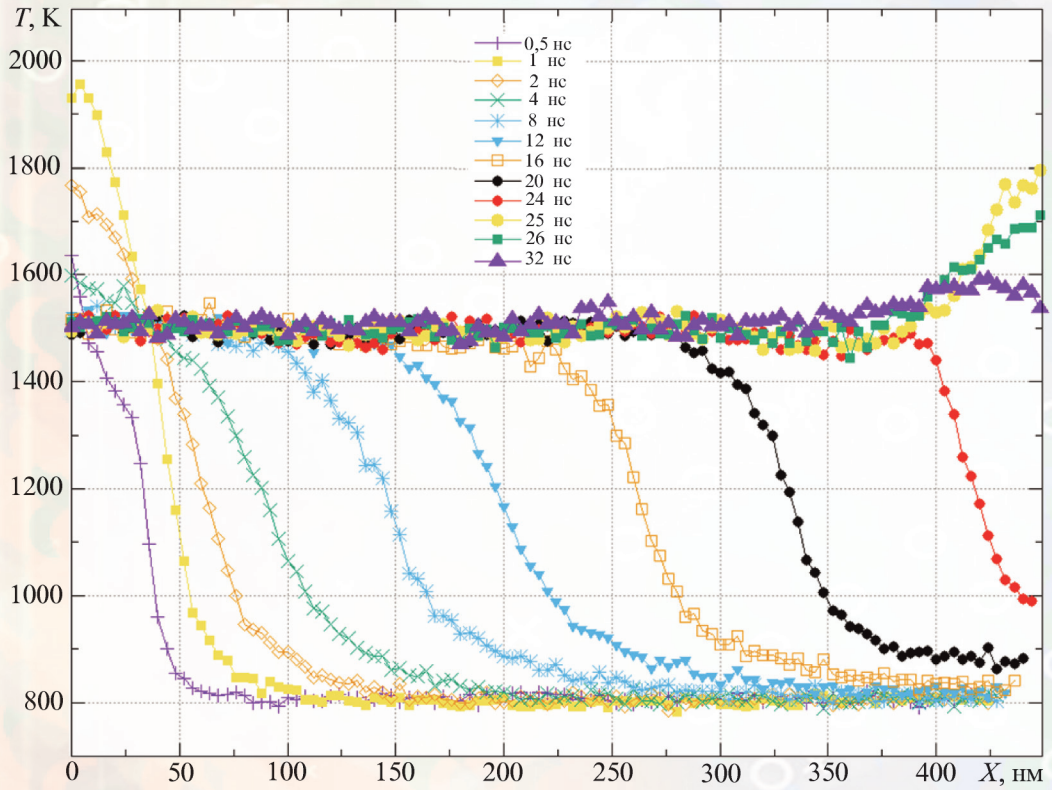


Рис. 4. Температурные профили, соответствующие последовательности моментов времени при продвижении волны горения в СБС системы Ti-Al

ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Издается с сентября 2010 г. DOI 10.17587/issn.2220-3397 ISSN 2220-3397

Журнал издается при поддержке Отделения математических наук РАН, Отделения нанотехнологий и информационных технологий РАН, МГУ имени М.В. Ломоносова, МГТУ имени Н.Э. Баумана

СОДЕРЖАНИЕ

Журнал зарегистрирован в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций. Свидетельство о регистрации ПИ № ФС77-38590 от 24 декабря 2009 г.

Журнал распространяется по подписке, которую можно оформить в подписных агентствах (индекс по Объединенному каталогу "Пресса России" — **22765**) или непосредственно в редакции (для юридических лиц).
Тел.: (499) 270-16-52.
Http://novtex.ru/prin/rus E-mail: prin@novtex.ru
Журнал включен в Российский индекс научного цитирования (РИНЦ) и Russian Science Citation Index (RSCI).
Журнал входит в Перечень научных журналов, в которых по рекомендации ВАК РФ должны быть опубликованы научные результаты диссертаций на соискание ученой степени доктора и кандидата наук.

THEORETICAL AND APPLIED SCIENTIFIC AND TECHNICAL JOURNAL

SOFTWARE ENGINEERING

PROGRAMMNAYA INGENERIA

Vol. 13
N 10
2022

Published since September 2010 **DOI 10.17587/issn.2220-3397** **ISSN 2220-3397**

CONTENTS

Information about the jornal is available online at:
http://novtex.ru/prin/eng e-mail:prin@novtex.ru

УДК 004.2+004.4
DOI: 10.7587/prin.13.471-482

В. В. Корнеев, д-р техн. наук, проф. гл. науч. сотр., korv@rdi-kvant.ru,
ФГУП «Научно-исследовательский институт "Квант"», Москва

Маршрутизация в коммуникационной среде вычислительной системы с распределенной разделяемой памятью и синхронизацией на базе FE-битов

Рассмотрена реализация распределенной разделяемой памяти с синхронизацией на базе FE-битов. Предложена арифметическая маршрутизация в коммуникационной среде вычислительной системы с распределенной разделяемой памятью, устраняющая взаимовлияние параллельных программ, исполняемых в разных подсистемах, и снижающая энергозатратность коммутаций в целом.

Ключевые слова: *архитектура вычислительной системы с распределенной разделяемой памятью, модель параллельного программирования, координатная адресация, энергоэффективная маршрутизация, арифметическая маршрутизация*

Введение

Существующие суперкомпьютеры ориентированы на достижение высокой производительности при проведении больших вычислений, например, вычислений с большими плотными матрицами. Однако их производительность падает до неприемлемого уровня при работе с большими данными. К их числу относятся, например, данные, представленные большими графами, в том числе потоками таких данных [1, 2], решения по выявлению фактов из больших потоков данных, поступающих из большого числа разнообразных источников.

Для работы с большими данными с приемлемой производительностью необходимы новые архитектурные подходы. Их поиск ведется среди как специализированных архитектур, так и расширения возможностей существующих мультитредовых архитектур. Так, для решения задач с большими графами предложена специализированная архитектура [3, 4] на базе алгоритмов библиотеки GraphBLAS [5], ориентированная на работу с большими разреженными матрицами, используемыми для представления графов. В предлагаемой архитектуре элементы разреженных матриц и элементы результирующей матрицы распределяются по процессорным элементам. В каждом процессорном элементе вычисляются частичные результаты из размещенных в них элементов исходных матриц. Эти результаты передаются в соответствующие процессорные элементы, в которых формируются элементы результирующей матрицы. В процессорных элементах используется аппаратный ускоритель — систолическая сортировка.

Более универсальная мультитредовая EMU-архитектура предложена в работе [6]. Ее особенностью является множество мультитредовых процессорных элементов, каждый из которых имеет "узкий" байтовый доступ к собственному блоку некешируемой памяти, что увеличивает пропускную способность памяти за счет большого числа блоков и обмена только необходимым числом байтов. Общее адресное пространство распределено по процессорным элементам, и тред, обращающийся к слову памяти, адрес которого в другом процессорном элементе, мигрирует в этот элемент. Наряду с этим используется также технология PCI по образованию разделяемой памяти, доступной всем процессорным элементам, что делает возможным размещение в ней программ, данных и констант, общих для всех тредов.

В работе [7] предложена архитектура PIUMA (*programmable integrated unified memory architecture*), которая обобщает EMU-архитектуру [6], добавляя в процессорном элементе к множеству мультитредовых ядер специализированные ускорители

(*memory offload engines*): канал прямого доступа к памяти (DMA) с возможностями копирования, сборки, рассылки (*copy*, *scatter*, *gather*) данных между блоками памяти, а также ускорители для работы с очередями (*queue engines*), ускорители-синхронизаторы (*collective engines*), обеспечивающие выполнение атомарных операций и барьерную синхронизацию. В отличие от EMU-архитектуры, распределение тредов по ядрам управляется программно без автоматической миграции тредов к обрабатываемым данным.

Хотя существуют отдельные оценки производительности рассмотренных архитектур, например, на задаче умножения разреженной матрицы на плотный вектор [7, 8], в целом неизвестно о реализации систем с этими архитектурами и, соответственно, об их возможности решать актуальные задачи с большими данными в приемлемое время. Вместе с тем во всех архитектурах делается упор на повышение эффективности работы с разделяемой памятью. Следует также отметить, что предлагаемая разделяемая память реализуется на базе традиционных коммуникационных сред, мало подходящих для интенсивных обращений к большим данным, в том числе в силу их энергозатратности.

Существующие суперкомпьютеры демонстрируют очевидный предел по потребляемой энергии, что делает проблематичным их экстенсивное развитие при сохранении используемых архитектурно-технических решений. Суперкомпьютер Frontier потребляет более 20 МВт при проведении вычислений с большими плотными матрицами, достигая производительности 1,102 Экзафлоп/с на 8 730 112 вычислительных ядрах [9]. Архитектурно-технические решения, принятые в суперкомпьютере Frontier, представляют достигнутый передний край, что подтверждается первым местом в соответствующем списке энергоэффективных компьютеров [10].

Коммуникационная среда Slingshot [11] вносит весомый вклад в достижение энергоэффективности суперкомпьютера Frontier. В Slingshot функции сети Ethernet реализуются без энергозатратных типовых решений этой сети с использованием контекстно-адресуемой памяти. В ней аппаратно встроены алгоритмы поиска наименее нагруженных маршрутов, обеспечения соглашения об уровне обслуживания, введения классов пакетов и другие алгоритмы, ранее опробованные в программных продуктах управления коммуникационными сетями суперкомпьютеров.

В настоящее время идет интенсивное исследование способов построения энергоэффективных отказоустойчивых коммуникационных сред с высокой пропускной способностью и малой латентностью. Например, в работе [12] рассмотрены различные алгоритмы маршрутизации, в том числе отказоустойчивой, устраняющей влияние сетевых отказов на исполнение прикладных программ. В большинстве коммуникационных сред современных суперкомпьютеров для маршрутизации пакетов используются таблицы, размещаемые для ускорения поиска соответствия в блоках энергозатратной ассоциативной памяти, объем и число которых также возрастают с увеличением числа вычислительных модулей. По этой причине интенсивно исследуются возможности создания арифметической маршрутизации [13], при которой маршрут задается самим адресом и определяется путем достаточно простых вычислений.

В целом для коммуникационной среды определяющим фактором эффективности служит граф связей — топология и алгоритм маршрутизации. Все остальные алгоритмы, направленные на устранение конфликтов, "укорочение" хвоста распределения задержек (*tail of latency distribution*), обеспечения качества обслуживания и отказоустойчивости, применимы, как правило, для всех топологий и маршрутизаций.

Часто используемой топологией служит Dragonfly, рекомендуемая в качестве основной в коммуникационной среде Slingshot, но в работе [11] отмечено, что существуют другие топологии с меньшей длиной среднего пути, но для них надо использовать более сложные и длительно выполняемые энергозатратные маршрутизации. В работе [14] показано преимущество топологии с меньшей длиной среднего пути по сравнению с другими топологиями, включая Dragonfly, при выполнении коллективных операций MPI и ряда тестовых программ, таких как NAS Parallel Benchmarks (NPB version 3.3.1 on MPI) и Graph 500.

В настоящей статье рассмотрена архитектура, базирующаяся на парадигме использования всего возможного параллелизма обработки [15—18]. Пользователь должен только указывать, какие вычисления можно проводить параллельными потоками над общей разделяемой памятью, сообразуясь только с выбранным алгоритмом. Такой подход позволяет создать максимальный поток обращений к памяти, присущий алгоритму. При необходимости прочитать ячейку общей памяти одним потоком, и только потом записать в нее новое значение другим потоком, пользователь полагает, что механизм разрешения конфликта реализуется аппаратными средствами управления доступом к памяти. В целом предлагаемая архитектура направлена на решение тех же задач, что и архитек-

туры EMU и PIUMA. Однако она использует для синхронизации тредов и реализации атомарных операций "умные" смарт-контроллеры блоков разделяемой памяти.

Для большого потока обращений к распределенной разделяемой памяти необходима энергоэффективная маршрутизация. В настоящей статье предложена арифметическая маршрутизация, которая применима в любых коммуникационных средах, в том числе с графами межмашинных связей Dragonfly и Л(N, v, 7) с числом вершин N и степенями вершин v. При диаметре 3 графы Л(N, v, 7) имеют минимально возможную длину среднего пути [15, 16]. Предложен алгоритм маршрутизации, обеспечивающий отказоустойчивое функционирование на базе возможности выбора альтернативных маршрутов.

Основы архитектуры для работы с большими данными

Экзафлопсные компьютеры строятся из вычислительных модулей (ВМ), каждый из которых имеет совокупность универсальных и специализированных вычислительных ядер, блок управления памятью (*Memory Management Unit* — MMU), смарт-контроллеры блоков памяти, сами блоки памяти и сетевые контроллеры коммуникационной среды, объединяющей все вычислительные модули [15, 17, 18]. Число смарт-контроллеров, собственно блоков памяти и сетевых контроллеров определяется исходя из необходимости обеспечения пропускной способности, минимизирующей простой ядер ВМ в ожидании завершения доступа к данным.

Распределенная разделяемая память суперкомпьютера состоит из совокупности блоков памяти, размещенных в каждом ВМ. Блоки распределенной разделяемой памяти, размещенные в ВМ, будем называть близкими для процессорных ядер этого ВМ, все остальные — удаленными. Число блоков памяти в каждом ВМ, с одной стороны, должно быть как можно больше, что обеспечивает увеличение доли обращений к близким блокам памяти. С другой стороны, следует иметь в виду, что их число ограничивается сложностью управления и конструкцией ВМ.

При инициации суперкомпьютера выполняется конфигурация глобального адресного пространства путем распределения адресного пространства по блокам памяти. Это распределение фиксируется в блоках управления памятью MMU, основной функцией которых является направление запросов к контроллерам блоков памяти с соответствующими адресами.

Каждое ядро при выполнении текущим тредом команды доступа к разделяемой памяти по чтению после выдачи обращения к памяти приостанавливает этот тред до получения ответа от памяти.

Обращение треда к памяти по чтению или по записи помещается в очередь необслуженных запросов к памяти в блоке управления памятью MMU. Далее MMU определяет, куда идет обращение — к близкому или удаленному блоку памяти другого ВМ.

В случае обращения к близкому блоку памяти это обращение посылается в соответствующий смарт-контроллер памяти.

Для сокращения трафика и ускорения работы с памятью может быть реализована модель программирования с локальными чтениями и локальными или удаленными записями [19]. Так описано, например, в работах [6, 20].

При обращении к близкому блоку по записи соответствующий запрос удаляется из очереди необслуженных запросов к памяти в блоке управления памятью MMU.

При обращении по чтению запрос из очереди необслуженных запросов в MMU удаляется после получения данных из блока памяти, передачи этих данных ждущему треду и перевода ждущего треда в состояние готовых к исполнению.

В случае обращения к удаленному блоку памяти по записи формируется обращение к сетевому контроллеру, который должен переслать в другой ВМ обращение соответствующему удаленному блоку памяти, удалив после этого запрос из своей очереди необслуженных запросов. Адресуемый ВМ, в котором размещен блок памяти, в который должна быть проведена запись данных, получив соответствующий адрес и сами данные по сети формирует запрос по записи в очереди необслуженных запросов MMU этого ВМ. Далее этот запрос передается в соответствующий смарт-контроллер памяти для выполнения и удаляется из очереди необслуженных запросов MMU. То есть с момента попадания запроса на запись в очередь необслуженных запросов MMU его исполнение не отличается от исполнения запроса на запись в близкий блок памяти.

Языком параллельного программирования, реализующим модель PRAM (*Parallel Random Access Model*), может служить расширение языка C [21] и Cilk [22]. Для порождения и завершения асинхронных тредов в Cilk используются три дополнительные функции: cilk_spawn, cilk_for, cilk_sync. Эти функции позволяют, соответственно:

- породить новый тред, исполняющий код, непосредственно следующий за cilk_spawn;

- выполнить порождение в цикле for совокупности тредов;
- завершить исполняющий тред и перейти к следующему оператору, если завершены все порожденные соответствующей функцией cilk_spawn треды.

При порождении тредов создается контекст треда, достаточный чтобы выполнить последующее завершение всех порожденных тредов. Собственно этот контекст используется в смарт-контроллере при выполнении соответствующей функции cilk_sync. Контекст треда, наряду с требуемыми ресурсами, обязательно включает адрес слова разделяемой памяти, в котором хранится число завершенных тредов. В контексте также есть переменная, содержащая число порожденных тредов. Слово памяти, в котором хранится число завершенных тредов, обнуляется при порождении тредов и увеличивается при достижении тредом завершения cilk_sync. При достижении значения, равного числу порожденных тредов, выполняется переход к следующему за cilk_sync оператору.

Если тред порождает тред или совокупность тредов, то каждый из них наследует контекст порождающего треда, расширенный указателями на этот тред и рядом параметров. Естественно, все порожденные треды должны быть завершены до порождающего. У порожденных тредов в контексте используется собственное слово для подсчета завершенных тредов. Уровень вложенности тредов ограничивается принятой структурой контекста.

Следует отметить, что в Cray XMT [23] расширение языка C для использования синхронизации на базе FE-битов реализовано как введение синхронизирующих переменных x$ и аппаратных функций (*generic functions*) для выполнения операций чтения и записи. Среди них отметим:

- purge x$ — присвоение FE-биту x$ значения empty;
- writeqr x$, *g* — запись в x$ значения *g*, если значение FE-бита x$ равно *q*, или ожидание записи, пока значение FE-бита не станет *q*; после записи значение FE-бита становится равным *r*;
- readqr x$ — чтение значения x$, если значение FE-бита x$ равно *q*, или ожидание чтения пока значение FE-бита не станет *q*; после чтения значение FE-бита становится равным *r*.

Смарт-контроллер памяти для каждого запроса независимо может работать в обычном режиме или в расширенном режиме с учетом механизма FE-битов.

Смарт-контроллер содержит память FE-битов слов памяти блоков, к которым он обеспечивает доступ. Размещение FE-битов слов в памяти, отдельной от самого блока памяти, имеет ряд преимуществ. Во-первых, позволяет исключить обращения к памяти при несоответствующем требуемому состоянии FE-бита слова памяти, во-вторых, позволяет выполнять операции с FE-битами без реального обращения к блокам памяти, и, в-третьих, позволяет использовать традиционный подход к построению блоков памяти и функционированию без использования FE-битов.

В расширенном режиме смарт-контроллер реализует задаваемую модель PRAM: PRAM-EREW (*exclusive-read exclusive-write*); PRAM-CREW (*concurrent-read exclusive-write*); PRAM-CRCW (*concurrent-read concurrent-write*) [24]. Однако все они, кроме PRAM-EREW требуют достижения обращения к слову блока памяти всеми порожденными тредами, что может повлечь неприемлемую потерю производительности (неявное введение барьера). Вопрос использования двух других моделей требует дополнительного исследования.

Если FE-бит не имеет требуемого значения, то обращение к памяти помещается в очередь ожидания. Элементы этой очереди содержат собственно обращение к памяти (команду чтения/записи, адрес слова, значения FE-битов до выполнения обращения и по его завершении, служебную информацию для работы в расширенном режиме, а также указатель на незавершенную команду MMU). Изменение состояния FE-бита слова инициирует обработку очереди ожидания с реализацией требуемой операции.

Если FE-бит имеет требуемое значение, то контроллер блока памяти выполняет требуемое чтение или запись и формирует заданное значение FE-бита.

Не вдаваясь в детали реализации контроллера памяти, отметим, что после завершения обращения к слову памяти должны выполняться поиск в очереди ожидания элементов, соответствующих этому слову памяти, и проверка возможности выполнения соответствующего обращения к памяти. Если таковое возможно, то обращение пускается на выполнение и соответствующий элемент удаляется из очереди. Естественно, описанные действия для своего ускорения требуют ассоциативного поиска в очереди ожидания.

Атомарные операции compare-and-swap (CAS), fetch-and-add, test-and-set выполняются как операции чтения с дополнительными действиями, для реализации которых используется арифметико-логическое устройство смарт-контроллера, необходимое также для cilk_sync. Такая реализация устраняет проблемы с обеспечением атомарности, исключая возможность прерывания хода выполнения действий, составляющих операцию.

Это обстоятельство исключает необходимость использования вместо CAS пары load-reserved (lr) и store-conditional (sc) [25].

Распределенный характер обращений в память служит гарантией высокой производительности, пропорциональной числу используемых смарт-контроллеров и блоков памяти вычислительных модулей.

Энергоэффективная маршрутизация

При использовании порядка 10^7 и более ВМ для исполнения программ с интенсивными обменами данными коммуникационная среда превращается в существенного потребителя энергии. Это вызывается, в том числе тем обстоятельством, что программы пользователей именуют процессы, приписывая им логические номера 0, 1, ..., $n-1$, где n — число используемых ВМ, и для установления соответствия между адресацией по логическим номерам процессов и физическими адресами ВМ, в которых эти процессы протекают, используются таблицы. Объем памяти, занимаемой ими, пропорционален квадрату числа ВМ в системе, так как таблица должна быть в каждом ВМ.

В большинстве коммуникационных сред современных суперкомпьютеров пакеты маршрутизируются таблицами, размещаемыми для ускорения поиска в блоках энергозатратной ассоциативной памяти, объем и число которых также возрастают с увеличением числа ВМ. По этой причине интенсивно исследуются возможности создания арифметической маршрутизации [13].

Среди алгоритмов арифметической адресации известна координатная адресация, которая применяется, в том числе и в вычислительных системах с Л(N, v, 4)-графом [15, 16], или многомерными кубическими структурами в качестве графов межмодульных связей, в которых адрес ВМ задается координатами по каждому направлению. При передаче элемента данных сравниваются значения координат адреса назначения и ВМ, в котором выполняется сравнение. Если значения всех координат равны, адресат достигнут. Среди направлений передачи с несовпавшими координатами выбирается направление, ведущее к уменьшению рассогласования. Таким образом обеспечивается эффективная адресация. Она используется в ряде суперкомпьютеров, например, фирмы CRAY.

Для вычислительных систем с Л(N, v, g)-графом межмодульных связей, где N — число вершин; v — степени вершин; g — обхват графа, предложена Д(z, m) — адресация [16], при которой адрес ВМ_i, $i \in \{0, \ldots, n-1\}$, также задается вектором $A_i = A_{i0}, \ldots, A_{i\,m-1}$. Каждый A_{ij}, $j = 0, \ldots, m-1$, принимает значение из множества $\{0, 1, \ldots, 2^z - 1\}$, $m \in \{1, 2, \ldots\}$, $z \in \{1, 2, \ldots\}$, n — число ВМ подсистемы, $mz \geqslant \lceil \log_2 n \rceil$, $\lceil x \rceil$ — наименьшее целое такое, что $x \leqslant \lceil x \rceil$. ВМ_i и ВМ_k, $i, k \in \{0, 1, \ldots, n-1\}$, принадлежат подсистеме $R_r(A_{i0}, \ldots, A_{i\,r-1})$ яруса r, если $A_{i\,r} \neq A_{kr}$, и $A_{ij} = A_{kj}$ для всех $j < r$, $r = 1, 2, \ldots, m$. $R_0()$ — подсистема яруса 0 (вся подсистема из n ВМ), $R_m(A_{i0}, \ldots, A_{im-1})$ — подсистема яруса m, состоящая из одного ВМ_i. Адресация ВМ должна удовлетворять следующим свойствам:

- ВМ каждой подсистемы $R_r(A_{i0}, \ldots, A_{ir-1})$ яруса r, $r = 1, 2, \ldots, m$, должны вместе с линиями связи между ними отображаться в связный подграф графа межмодульных связей;
- $R_0 \supseteq R_1(A_{i0}) \supseteq R_2(A_{i0}, A_{i1}) \supseteq \ldots \supseteq R_m(A_{i0}, \ldots, A_{im-1})$;
- $R_r(A_{i0}, \ldots, A_{ir-1}) = \bigcup_{j=0}^{d} R_{r+1}(A_{i0}, \ldots, A_{ir-1}, j)$,

$d = 2^z - 1$;

- $R_{r+1}(A_{i0}, \ldots, A_{ir-1}, j) \cap R_{r+1}(A_{i0}, \ldots, A_{ir-1}, k) = \varnothing$, $j \neq k$.

При Д(z, m)-адресации в каждом ВМ_i, $i \in \{0,1, \ldots, n-1\}$, должно храниться в путевой таблице только $m2^z$ номеров $p_1(0)$, $p_1(1)$, ..., $p_1(2^z-1)$, $p_2(0)$, $p_2(1)$, ..., $p_2(2^z-1)$, ..., $p_m(0)$, $p_m(1)$, ..., $p_m(2^z-1)$ выходных полюсов ВМ_i, принадлежащих кратчайшим путям из ВМ_i в соответственно подсистемы $R_1(0)$, $R_1(1)$, ..., $R_1(2^z-1)$, $R_2(A_{i0}, 0)$, ..., $R_2(A_{i0}, 2^z-1)$, ..., $R_m(A_{i0}, \ldots, A_{im-2}, 2^z-1)$. Кратчайший путь до подсистемы — это путь до ближайшего ВМ этой подсистемы.

Алгоритм маршрутизации сравнивает адресата пакета A_k с адресом ВМ, в котором он находится, и определяет старший, начиная с 0, не совпадающий разряд j, $j \in \{0, 1, \ldots, m-1\}$. Далее из путевой таблицы берется значение, соответствующее A_{kj}, которое задает номер линии связи для передачи пакета. Если адресат пакета и адрес ВМ, в котором он находится, совпадают, пакет доставлен. Как видно, отличие от координатной адресации заключается в том, что ищется старший не совпадающий разряд, а не любой. Кроме того, необходимо нахождение в путевой таблице номера линии связи, а не прямое его определение номером разряда рассогласования.

Представленная маршрутизация на основе Д(z, m)-адресации требует при программировании таблицы соответствия номеров и адресов в каждом ВМ, так как программирование ведется в терминах номеров процессов, а передача пакетов сообщений между процессами выполняется на основании адресов ВМ. Представим алгоритм маршрутизации, в котором используется Д(1, m)-

адресация с адресами, задаваемыми номерами ВМ в двоичном представлении m разрядами.

Создадим Д(1, m)-адресацию на выделенной связной подсистеме из N ВМ, значение m определится в ходе построения адресации. Итак, будем полагать, что есть алгоритм, строящий связную подсистему из виртуальных ВМ, с производительностью выше заданного порога и линиями связи между ВМ, с пропускной способностью выше заданного порога, что определяется соглашением о качестве облуживания.

Пусть имеем граф с N вершинами и ребрами, помеченными их пропускной способностью. Находим для этого графа паросочетание вершин (*matching*) с максимальной суммой меток выбранных ребер [23]. Пусть для графа на рис. 1, $N = 10$, получится паросочетание, приведенное на рис. 2. Вершины, инцидентные выбранным ребрам, соответствуют двухвершинным подсистемам нижнего уровня, а каждая вершина, не инцидентная выбранным ребрам, образует одновершинную подсистему нижнего уровня. В целом выбор делается так, чтобы получить максимальную суммарную пропускную способность линий связи подсистем.

Далее двухвершинные подсистемы стягиваются в одну вершину с сохранением инцидентных им ребер (рис. 3).

Находим для этого графа паросочетание вершин (*matching*) с максимальной суммой выбранных ребер. И повторяем последовательность этих действий, пока не получим двухвершинный граф. Соответствующие шаги алгоритма иллюстрируются на рис. 4—6.

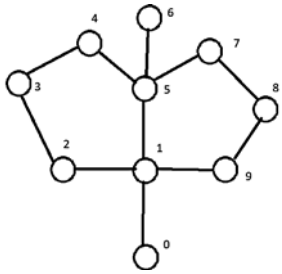


Рис. 1. Граф межмодульных связей выделенной подсистемы

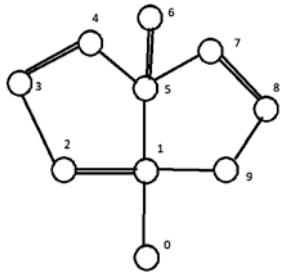


Рис. 2. Сформированные паросочетания на шаге 1 алгоритма

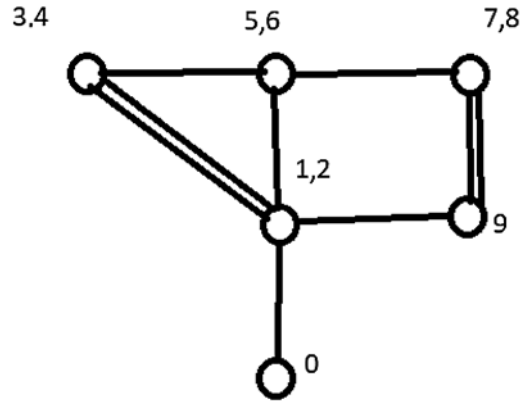


Рис. 3. Сформированные паросочетания на шаге 2 алгоритма

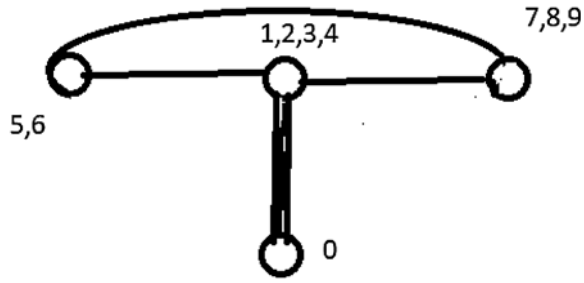


Рис. 4. Сформированные паросочетания на шаге 3 алгоритма

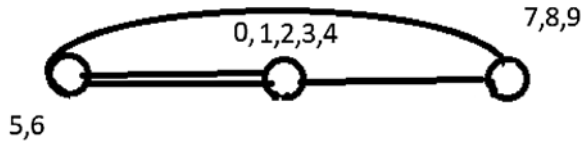


Рис. 5. Сформированные паросочетания на шаге 4 алгоритма

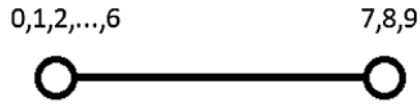


Рис. 6. Сформированные паросочетания на шаге 5 алгоритма

Число сделанных шагов определяет параметр m. В рассматриваемом примере их 5. Таким образом создана Д(1, 5)-адресация, приведенная на рис. 7.

При задании адресов одноэлементному подмножеству всегда приписывается 0, что обеспечивает обязательное присутствие номера 0 с адресом 0...0 с нулями во всех m разрядах адреса.

Далее в каждом ВМ формируются путевые таблицы $T[i]$, $i = 0, ..., m - 1$. По завершении элемент $T[i]$, $i = 0, ..., m - 1$:

1) если его значение больше 0, то задает номер связи, по которой пакет должен быть передан из ВМ, чей адрес отличается в разряде i и совпадает во всех предшествующих 0, 1, ..., $i - 1$ разрядах с адресатом пакета;

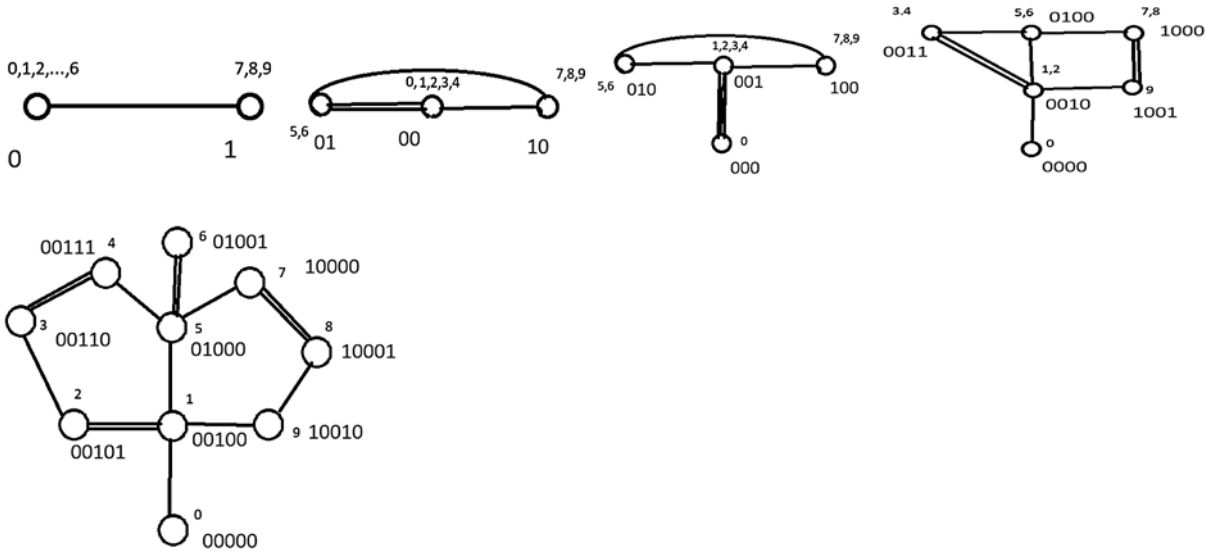


Рис. 7. Сформированная Д(1, 5)-адресация

2) если имеет ненулевое отрицательное значение, то его модуль задает адрес перенумерованного ВМ, и должен быть записан в пакет, как адресат пакета, после чего пакет заново маршрутизируется;

3) если имеет нулевое отрицательное значение, то задает отсутствие такого адреса.

Формирование путевых таблиц начинается с заполнения их нулевыми отрицательными значениями. Это необходимо для выделения несуществующих адресов.

Также формируются вспомогательные таблицы $H[i, j]$, $i = 0, ..., m - 1$, $j = 0, ..., v - 1$, содержащие минимальное расстояние (число передач) до соответствующей адресной подсистемы, исходя из ВМ по связи $j + 1$. Исходно — максимально возможное расстояние.

Во всех ВМ генерируются сообщения с полями:

- номер ВМ;
- адрес ВМ;
- количество передач (исходно равно 1);
- максимальное количество передач;
- номер связи, по которой пришло (исходно равен 0).

Сообщение посылается по всем линиям связи, кроме той, по которой поступило в ВМ. Если сообщение сгенерировано в ВМ, номер связи 0, сообщение посылается по всем линиям связи.

Получив сообщение, ВМ сравнивает свой адрес с адресом в сообщении. Находится старший разряд i, $i \in \{0, ..., m - 1\}$, несовпадения. Сравнивается значение в таблице $H[i, j - 1]$, где j равно номеру связи, $j \in \{1, 2, ..., v\}$, по которой сообщение пришло в ВМ, и число передач сообщения. Если число передач меньше значения в таблице $H[i, j - 1]$, то в $H[i, j - 1]$ заносится число передач из сообщения. Тем самым сколько бы сообщений не прошло, даже одно, будет сформирован участок маршрута к адресной подсистеме $A_0...A_i$, $i \in \{0, ..., m - 1\}$. Если пройдут все сообщения, то $H[i, j - 1]$ будет содержать длину кратчайшего маршрута из рассматриваемого ВМ до адресной подсистемы $A_0...A_i$, $i \in \{0, ..., m - 1\}$, включающего связь с номером j. Если такого маршрута не окажется, то его длина будет максимальной, т. е. исходно заданной.

Кроме того, адрес в сообщении сравнивается с n — числом ВМ в подсистеме. Если адрес больше $n - 1$, то он запоминается в списке перенумерумых ВМ. В каждом ВМ список сортируется по возрастанию, таким образом, все ВМ сформируют одинаковые списки и определится число перенумеруемых ВМ. Так, для адресации, показанной на рис. 7, в список попадут ВМ с адресами 10000, 10001, 10010.

Если не превышено максимальное число передач сообщения, оно транслируется далее по всем линиям связи, кроме той, по которой поступило в ВМ.

После завершения формирования вспомогательных таблиц $H[i, j]$, $i = 0, ..., m - 1$, $j = 0, ..., v - 1$, создаются путевые таблицы $T[i]$, $i = 0, ..., m - 1$. В каждом ВМ выбирается в каждой строке i, $i = 0, ..., m - 1$, столбец таблицы $H[i, j]$, $i = 0, ..., m - 1$, $j = 0, ..., v - 1$, с наименьшим, не равным максимальному значению расстоянием до адресуемой подсистемы, и соответствующий номер столбца становится значением $T[i]$, $i = 0, ..., m - 1$. Если вся строка i, $i = 0, ..., m - 1$, таблицы имеет исходные максимальные значения, то $T[i]$, получает значение -0.

В каждом ВМ становится возможным определить номера ВМ, адреса которых будут не совпа-

дать с их номерами. Если адрес ВМ меньше n, то анализируются элементы путевой таблицы. Если обнаруживается $T[i]$, $i \in \{0, ..., m-1\}$, равный -0, то номера, начинающиеся на $A_0A_1...A_{i-1}$, кроме собственно номера ВМ, определяются как отсутствующие. Так, для адресации, показанной на рис. 7, в ВМ с адресом 00000 $T[3]$ и $T[4]$ равны -0, что обнаруживает отсутствие номеров 00001, 00010, 00011.

Далее каждый ВМ по восходящему и нисходящему корневым деревьям с корнем 0 ... 0 передает обнаруженные отсутствующие номера и принимает эти номера, сортируя их по возрастанию. Так как число таких номеров известно, то процесс в каждом ВМ завершается их принятием. Тем самым в каждом ВМ формируется соответствие отсутствующих номеров и перенумерумых ВМ. Для адресации, показанной на рис. 7, 00001—10000, 00010—10001, 00011—10010.

Формирование адресации завершается в каждом ВМ выделением соответствующего ему подсписка переадресации отсутствующих адресов и сопоставленных перенумерумых ВМ.

Алгоритм маршрутизации модифицируется. Если обнаруживается $T[i]$, $i \in \{0, ..., m-1\}$, равный -0, то из подсписка переадресации отсутствующих адресов выбирается номер, сопоставленный адресату сообщения. Далее этот номер заносится в сообщение в качестве его адресата. Полученное сообщение подвергается маршрутизации для достижения перенумерованного ВМ.

Представленные алгоритмы формирования адресации и маршрутизации гарантируют пользователю предоставление для выполнения параллельной программы ВМ с номерами 0, 1, ..., $n-1$. Если автоматическая переадресация в алгоритме маршрутизации признается пользователем излишней, он может проводить подмену адресов в своей программе, используя данные из соответствующих таблиц.

Для обеспечения отказоустойчивости и обхода перегруженных участков маршрута алгоритм маршрутизации может использовать данные таблиц $H[i, j]$, $i = 0, ..., m-1$, $j = 0, ..., v-1$, содержащие сведения об альтернативных маршрутах до адресуемых подсистем.

Следует отметить, что на одном и том же графе связей подсистемы, в зависимости от выбранных паросочетаний вершин, может быть сформирована адресация с меньшим числом уровней. Так, для подсистемы, изображенной на рис. 1, может быть сформирована адресация, показанная на рис. 8.

Примечательно, что при меньшем значении $m = 4$ понадобится только одна переадресация: адрес 0101 заменяется на 1010. Эта адресация получается, если на шаге 1 выбраны паросочетания (0, 1), (3, 4), (5, 6), (7, 8), на шаге 2 — ((3, 4), 2), ((0, 1), (5, 6)), ((7, 8), 9), на шаге 3 — (((0, 1), (5, 6)), ((3, 4), 2))) и, наконец, на шаге 4 — ((((3, 4), 2), ((0, 1), (5, 6))), ((7, 8), 9)).

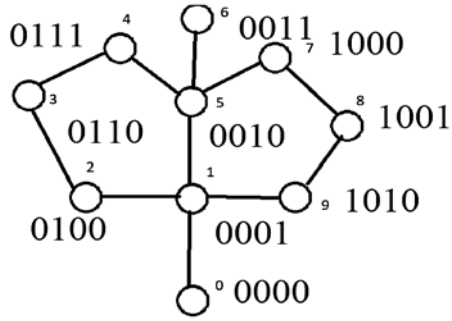


Рис. 8. Сформированная Д(1, 4)-адресация

Формирование подсистем с заданной адресацией

Рассмотренный алгоритм маршрутизации вписывается в архитектуру и организацию функционирования суперкомпьютера с разделяемой памятью с синхронизацией на базе FE-битов слов памяти [15—18]. Исходно весь суперкомпьютер рассматривается как совокупность всех ресурсов [26]. Для исполнения каждой программы выделяется подсистема, состоящая из затребованного числа виртуальных ВМ, соединяющих их линий и других ресурсов. Кратко, со ссылкой на более подробное описание, представим децентрализованный распределенный параллельный алгоритм построения подсистем, несколько модифицированный по сравнению с представленным в работе [16].

Построение подсистемы в суперкомпьютере с Л(N, v, g)-графом межмашинных связей инициируется из любого ВМ. В ВМ формируется сообщение с полями:

- тип сообщения "распространение";
- число n ВМ, которое требуется включить в строящуюся подсистему этим сообщением; $n \in \{1, ..., N\}$;
- тег подсистемы;
- соглашение о качестве обслуживания;
- номер линии связи, по которой сообщение поступило, изначально устанавливается 0.

Будем полагать, что в каждом ВМ имеется локальная очередь для сообщений с тегом подсистемы, помещаемых в нее по поступлении по линии связи или из самого ВМ, с внесением в поле "номер линии связи" номера линии связи, по которой сообщение поступило. Кроме того, имеется совокупность локальных переменных ВМ, определенных далее по мере использования.

Итак, в ВМ запускается программа построения подсистемы, в очереди которой находится сформированное сообщение типа "распространение".

По получении сообщения типа "распространение" определяется, порожден или нет виртуальный ВМ подсистемы с тегом сообщения. Если не порожден, то выполняются действия по порождению.

Если при обработке сообщения типа "распространение" локальная проверка устанавливает возможность выполнения соглашения о качестве обслуживания, то порождается виртуальный ВМ с ресурсами, выделенными по соглашению о качестве обслуживания. Номер линии связи, по которой сообщение поступило, записывается в локальную переменную ВМ up (путь к $ВМ_0$). Если номер линии связи, по которой сообщение поступило, равен 0, то ВМ получает в этой подсистеме локальный номер 0. В иных случаях номер — максимально возможное число. Поле "количество ВМ" уменьшается на 1, его значением становится $n - 1$. В порожденном виртуальном ВМ формируется локальный вектор q_p, (q_p исходно равен 1), $p = 1, ..., v$, где p — номер линии связи. Если номер s линии связи, по которой сообщение поступило, больше 0, то $q_s \leftarrow 0$.

Если после выполнения перечисленных выше действий по порождению ВМ поле "количество ВМ" имеет значение $n = 0$, то тип сообщения "распространение" заменяется на "завершение". В поле "количество ВМ" заносится 1. Количество ВМ поддерева, корнем которого он служит [15, 16], устанавливается равным 1 в локальной переменной τ, $\tau \leftarrow 1$. Сообщение типа "завершение" отправляется по линии связи, по которой сообщение поступило (в случае 0 в ВМ, инициировавший построение подсистемы).

Если $n - 1 \geqslant 1$, то τ получает значение n, $\tau \leftarrow n$, и $n - 1$ делится между открытыми линиями связи этого ВМ с $q_p = 1$, $p = 1, ..., v$. Для передачи по каждой открытой линии связи на основе полученного формируются сообщения типа "распространение", в которых указано выделенное линии количество ВМ. Это количество ВМ также суммируется в q_p, где p — номер очередной рассматриваемой открытой линии связи.

Если ВМ уже есть, и пришло сообщение типа "распространение" на продолжение построения подсистемы из этого ВМ в силу того, что где-то не включено в подсистему запрошенное количество ВМ, то анализируется q_p, $p = 1, ..., v$, этого ВМ. Если все значения $q_p = 0$, $p = 1, ..., v$, то тип сообщения меняется на "тупик", и оно передается в ВМ из которого поступило. Если хотя бы одно q_p, $p = 1, ..., v$, этого ВМ больше 1, то обработка сообщения откладывается до момента, пока все q_p, $p = 1, ..., v$, не получат значения 1 или 0. То есть в ВМ ожидается ответ на все посланные сообщения типа "распространение". По получении последнего ответа и формировании актуального вектора q_p, $p = 1, ..., v$, этого ВМ локальная переменная $\tau \leftarrow \tau + n$. Количество n делится между открытыми линиями связи этого ВМ с $q_p = 1$, $p = 1, ..., v$. Для передачи по каждой открытой линии связи на основе полученного формируются сообщения типа "распространение", в которых указано выделенное линии количество ВМ. Это количество ВМ также суммируется в q_p, p — номер рассматриваемой линии связи.

Если устанавливается отсутствие возможности выполнения соглашения, то формируется сообщение с типом "ресурсный отказ", отправляемое по линии связи, по которой сообщение поступило (в случае 0 ВМ, инициировавший построение подсистемы, получает отказ).

При поступлении в ВМ сообщения типа "ресурсный отказ" номер s линии, по которой оно поступило, определяет $q_s \leftarrow q_s - n$, переменная $\tau \leftarrow \tau - n$, где n — значение поля "количество ВМ" сообщения. Если $q_s = 1$, то q_s обнуляется $q_s \leftarrow 0$. Тем самым линия связи закрывается. Тип сообщения меняется на "распространение".

При поступлении в ВМ сообщения типа "завершение" номер s линии, по которой оно поступило, определяет $q_s \leftarrow q_s - n$, где n — значение поля "количество ВМ" сообщения. Если все q_p, $p = 1, ..., v$, имеют после этого значения 1 или 0, то получены подтверждения на включение в подсистему запрошенного количества ВМ. Анализируется значение локальной переменной up. Если up не равно 0, то создается сообщение типа "завершение", в котором поле "количество ВМ" равно локальной переменной τ, то сообщение передается по линии, определяемой локальной переменной up.

Если значение up равно 0, то достигнут корневой $ВМ_0$. Построение подсистемы завершено. В каждом ВМ сформированы локальные переменные, необходимые для задания номеров ВМ и маршрутизации по восходящему и нисходящему корневым деревьям [15, 16]:

- локальная переменная τ задает количество ВМ корневого покрывающего поддерева, корнем которого служит этот ВМ, подсистемы с корнем $ВМ_0$;
- локальная переменная up служит направлением к корню;
- компоненты локального вектора q_p, $p = 1, ..., v$, равные 1, задают направления от корня.

При инициации суперкомпьютера строится подсистема, включающая все ВМ, на которых задается Д(1, m)-адресация. Номер ВМ служит префиксом адресов памяти блоков памяти, размещен-

ных в этом ВМ. Длина префикса $m \geqslant \lceil \log_2 N \rceil$. При использовании N порядка 10^7, $m \geqslant 24$.

Таким образом, адрес разделяемой памяти суперкомпьютера состоит, по крайней мере, из трех полей: номер ВМ, номер блока памяти ВМ, адрес в блоке памяти. При этом номер ВМ определяется в ходе формирования адресации.

Построение Д(1, m)-адресации выполняется на ВМ подсистемы децентрализованным распределенным параллельным алгоритмом с использованием алгоритма нахождения паросочетаний [23] в режиме эмуляции графа межмодульных связей.

Заключение

Представленная в статье организация работы с распределенной разделяемой памятью направлена на поддержку повышения производительности параллельного программирования и снижение сложности и энергозатратности исполнения программ. Для исполнения параллельной программы формируется подсистема из виртуальных вычислительных модулей с номерами в интервале 0, ..., $n - 1$, где n — число запрошенных программой ВМ. Для синхронизации легких тредов программ используются FE-биты слов разделяемой памяти. Смарт-контроллеры содержат отдельную память FE-битов слов блоков памяти и для каждого запроса независимо могут работать в обычном режиме или в расширенном режиме с учетом механизма FE-битов.

Предложенная арифметическая маршрутизация при использовании топологии с минимальным средним расстоянием устраняет взаимовлияние параллельных программ, исполняемых в разных подсистемах, и снижает энергозатратность коммутаций в целом.

Список литературы

1. **Kogge P. M.** Graph Analytics: Complexity, Scalability, and Architectures // 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017. P. 1039—1047. DOI: 10.1109/IPDPSW.2017.176.

2. **Harrod W.** Advanced Graphical Intelligence Logical Computing Environment (AGILE) // ISC 2022 IXPUG Workshop. June 2, 2022. Hamburg Germany. URL: https://www.iarpa.gov/images/PropsersDayPDFs/AGILE/AGILE_-_ISC_2022_Harrod_Updated.pdf

3. **Song W. S., Gleyzer V., Lomakin A., Kepner J.** Novel graph processor architecture, prototype system, and results // 2016 IEEE High Performance Extreme Computing Conference (HPEC), 2016. P. 1—7.

4. **Song W. S.** Processor for large graph algorithm computations and matrix operations. United States Patent US 8751556B2.

5. **Kepner J., Aaltonen P., Bader D.** et al. Mathematical Foundations of the GraphBLAS. arXiv:1606.05790v2.

6. **Dysart T., Kogge P. M., Deneroff M.** et al. Highly Scalable Near Memory Processing with Migrating Threads on the Emu System Architecture // 6th Workshop on Irregular Applications: Architecture and Algorithms (IA3). 2016. P. 2—9. DOI: 10.1109/IA3.2016.007.

7. **Aananthakrishnan S., Ahmed N. K., Cave V.** et al. PIUMA: programmable integrated unified memory architecture. arXiv:2010.06277.

8. **Belviranli M. E., Lee S., Vetter J. S.** Designing Algorithms for the EMU Migrating-threads-based Architecture // 2018 IEEE High Performance Extreme Computing Conference (HPEC), 2018. P. 1—7. DOI: 10.1109/HPEC.2018.8547571.

9. **The 59th** edition of the TOP500. URL: https://www.top500.org/lists/top500/2022/06/highs/

10. **GREEN500** LIST — JUNE 2022. URL: https://www.top500.org/lists/green500/list/2022/06/

11. **De Sensi D., Di Girolamo S., McMahon K. H.** et al. An In-Depth Analysis of the Slingshot Interconnect. arXiv:2008.08886v1 [cs.DC] 20 Aug 2020.

12. **Vigneras P., Quintin J. N.** The BXI routing architecture for exascale supercomputer // The Journal of Supercomputing. 2016. Vol. 72, No. 12. P. 4418—4437. DOI: 10.1007/s11227-016-1755-2.

13. **Concatto C., Pascual J. A., Navaridas J.** et al. A CAM-Free Exascalable HPC Router for Low-Energy Communications // Springer International Publishing AG, part of Springer Nature 2018/ M. Berekovic et al. (Eds.). ARCS 2018. LNCS 10793. P. 99—111. DOI: 10.1007/978-3-319-77610-1_8.

14. **Deng Y., Guo M., Ramos A. F., Huang X., Xu Zh., Liu W.** Optimal Low-Latency Network Topologies for Cluster Performance Enhancement. arXiv:1904.00513v1 [cs.NI].

15. **Корнеев В. В.** Параллельное программирование // Программная инженерия. 2022. Том 13, № 1. С. 3—16. DOI: 10.17587/prin.13.3-16.

16. **Корнеев В. В.** Архитектура вычислительных систем с программируемой структурой. Новосибирск: Наука, 1985. 166 с.

17. **Корнеев В. В.** Модель программирования и архитектура экзафлопсного суперкомпьютера // Открытые системы. СУБД. 2014. № 10. С. 20—22.

18. **Елизаров С. Г., Лукьянченко Г. А., Корнеев В. В.** Технология параллельного программирования экзафлопсных компьютеров // Программная инженерия. 2015. № 7. С. 3—10.

19. **Hoffmann H., Wentzlaff D., Agarwal A.** Remote Store Programming: A Memory Model for Embedded Multicore // Proceedings of the 5th international conference on High Performance Embedded Architectures and Compilers, HiPEAC 2010, Jan 2010. P. 3—17. DOI: 10.1007/978-3-642-11515-8_3.

20. **Davidson S., Taylor M. B., Dreslinski R. G.** et al. The Celerity Open-Source 511-Core RISC-V Tiered Accelerator Fabric: Fast Architectures and Design Methodologies for Fast Chips // IEEE Micro March/April 2018. Vol. 38, No. 2. P. 30-41. DOI: 10.1109/MM.2018.022071133.

21. **Wen X., Vishkin U.** FPGA-Based Prototype of a PRAM-On-Chip Processor // Proceedings of the 5th conference on Computing frontiers. New York, NY, USA: ACM, 2008. P. 55—66. DOI: 10.1145/1366230.1366240.

22. **Blumofe R. D., Joerg C. F., Kuszmaul B. C.** et al. Cilk: an efcient multithreaded runtime system // Proceedings of the fifth ACM SIGPLAN symposium on Principles and practice of parallel programming, ser. PPOPP'95. New York, NY, USA: ACM, 1995. P. 207—216. URL: DOI: 10.1145/209936.209958.

23. **Feo J.** Dataflow on Cray XMT. URL: https://www.youtube.com/watch?v=5nj1Ql0Eo5k&t=1266s

24. **Akl S. G.** Design and analysis of parallel algorithms. Prentice Hall, 1989. 412 p.

25. **Patterson D., Hennessy J. L.** Computer Organization and Design RISC-V Edition: The Hardware Software Interface. Morgan Kaufmann, 2017. 696 p.

26. **Николаев Д. С., Корнеев В. В.** Использование механизмов контейнерной виртуализации в высокопроизводительных вычислительных комплексах с системой планирования заданий SLURM // Программная инженерия. 2017. Том 8, № 4. С. 147—160. DOI: 10.17587/prin.8.147-160.

Routing in a Communication Fabric of a Computing System with Distributed Shared Memory and Synchronization based on FE¯Bits

V. V. Korneev, korv@rdi-kvant.ru,
FSUE R&D Institute "Kvant", Moscow, 125438, Russian Federation

Corresponding author:

Victor V. Korneev, Principal Researcher,
FSUE R&D Institute "Kvant", Moscow, 125438, Russian Federation
E-mail: korv@rdi-kvant.ru

Received on September 13, 2022
Accepted on September 29, 2022

This article discusses an architecture based on the paradigm of using all possible processes parallelism. The user should only specify which calculations can be performed in parallel threads over shared memory, conforming only to the selected algorithm. This allows you to create the maximum flow of memory accesses inherent in the algorithm. If necessary, read, and only then write a new value instead to the corresponding shared memory cell, the user believes that the conflict resolution mechanism is implemented by hardware memory access control. In general, the proposed architecture is aimed at solving the same problems as the EMU and PIUMA architectures, but uses "smart" controllers of shared memory blocks to synchronize threads and implement atomic operations.

For a large flow of accesses to distributed shared memory, energy-efficient routing is necessary. This paper proposes arithmetic routing, which is applicable in any communication fabrics, including with graphs of Dragonfly and graphs with the minimum possible length of the middle path and with the same number of vertices N and degrees of vertices v. An addressing and routing algorithm is proposed that provides energy-efficient access to distributed shared memory. Routing enables fault-tolerant operation based on the choice of alternative routes.

Keywords: *architecture of a computing system with distributed shared memory, parallel programming model, coordinate addressing, energy-efficient routing, arithmetic routing*

For citation:

Korneev V. V. Routing in a Communication Fabric of a Computing System with Distributed Shared Memory and Synchronization based on FE-Bits, *Programmnaya Ingeneria,* 2022, vol. 13, no. 10, pp. 471—482.

DOI: 10.17587/prin.13.471-482

References

1. **Kogge P. M.** Graph Analytics: Complexity, Scalability, and Architectures, *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW),* 2017, pp. 1039—1047. DOI: 10.1109/IPDPSW.2017.176.

2. **Harrod W.** Advanced Graphical Intelligence Logical Computing Environment (AGILE). ISC 2022 IXPUG Workshop. June 2, 2022. Hamburg Germany, available at: https://www.iarpa.gov/images/PropsersDayPDFs/AGILE/AGILE_-_ISC_2022_Harrod_Updated.pdf

3. **Song W. S., Gleyzer V., Lomakin A., Kepner J.** Novel graph processor architecture, prototype system, and results, *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, 2016, pp. 1—7.

4. **Song W. S.** Processor for large graph algorthm computations and matrix operations. United States Patent US 8751556B2.

5. **Kepner J., Aaltonen P., Bader D.** et al. Mathematical Foundations of the GraphBLAS. arXiv:1606.05790v2.

6. **Dysart T., Kogge P. M., Deneroff M.** et al. Highly Scalable Near Memory Processing with Migrating Threads on the Emu System Architecture, *6th Workshop on Irregular Applications: Architecture and Algorithms (IA3),* 2016, pp. 2—9. DOI: 10.1109/IA3.2016.007.

7. **Aananthakrishnan S., Ahmed N. K., Cave V.** et al. PIUMA: programmable integrated unified memory architecture. arXiv:2010.06277.

8. **Belviranli M. E., Lee S., Vetter J. S.** Designing Algorithms for the EMU Migrating-threads-based Architecture, *2018 IEEE High Performance Extreme Computing Conference (HPEC),* 2018, pp. 1—7, DOI: 10.1109/HPEC.2018.8547571.

9. **The 59th** edition of the TOP500, available at: https://www.top500.org/lists/top500/2022/06/highs/

10. **GREEN500** LIST — JUNE 2022, available at: https://www.top500.org/lists/green500/list/2022/06/

11. **De Sensi D., Di Girolamo S., McMahon K. H.** et al. An In-Depth Analysis of the Slingshot Interconnect. arXiv:2008.08886v1 [cs.DC] 20 Aug 2020.

12. **Vigneras P., Quintin J. N.** The BXI routing architecture for exascale supercomputer, *The Journal of Supercomputing,* 2016, vol. 72, no. 12. pp. 4418—4437. DOI: 10.1007/s11227-016-1755-2.

13. **Concatto C., Pascual J. A., Navaridas J.** et al. A CAM-Free Exascalable HPC Router for Low-Energy Communications, *Springer International Publishing AG, part of Springer Nature 2018/* M. Berekovic et al. (Eds.), ARCS 2018, LNCS 10793, pp. 99—111. DOI: 10.1007/978-3-319-77610-1_8.

14. **Deng Y., Guo M., Ramos A. F.** et al. Optimal Low-Latency Network Topologies for Cluster Performance Enhancement. arXiv:1904.00513v1 [cs.NI].

15. **Korneev V. V.** Parallel programming, *Programmnaya Ingeneria*, 2022, vol. 13, no. 1, pp. 3—16. DOI: 10.17587/prin.13.3-16 (in Russian).

16. **Korneev V. V.** *Architecture of computer systems with a programmable structure*, Novosibirsk, Nauka, 1985, 166 p. (in Russian).

17. **Korneev V. V.** Programming model and architecture of an exaflops supercomputer, *Otkrytye sistemy, SUBD*, 2014, no. 10, pp. 20—22 (in Russian).

18. **Elizarov S. G., Lukyanchenko G. A., Korneev V. V.** Technology of parallel programming of exaflops computers, *Programmnaya Ingeneria*, 2015, no. 7, pp. 3—10 (in Russian).

19. **Hoffmann H., Wentzlaff D., Agarwal A.** Remote Store Programming: A Memory Model for Embedded Multicore. *Proceedings of the 5th international conference on High Performance Embedded Architectures and Compilers, HiPEAC,* 2010, Jan 2010, pp. 3—17. DOI:10.1007/978-3-642-11515-8_3.

20. **Davidson S., Taylor M. B., Dreslinski R. G.** et al. The Celerity Open-Source 511-Core RISC-V Tiered Accelerator Fabric: Fast Architectures and Design Methodologies for Fast Chips, *IEEE Micro,* 2018, vol. 38, no. 2, pp. 30—41. DOI: 10.1109/MM.2018.022071133.

21. **Wen X., Vishkin U.** FPGA-Based Prototype of a PRAM-On-Chip Processor, Proceedings of the 5th conference on Computing frontiers, New York, NY, USA, ACM, 2008 pp. 55—66. DOI: 10.1145/1366230.1366240.

22. **Blumofe R. D., Joerg C. F., Kuszmaul B. C.** et al. Cilk: an efcient multithreaded runtime system, *Proceedings of the fifth ACM SIGPLAN symposium on Principles and practice of parallel programming, ser. PPOPP'95,* New York, NY, USA, ACM, 1995, pp. 207—216. URL: http://doi.acm.org/10.1145/209936.209958.

23. **Feo J.** Dataflow on Cray XMT. URL: https://www.youtube.com/watch?v=5nj1Ql0Eo5k&t=1266s

24. **Akl S. G.** *Design and analysis of parallel algorithms,* Prentice Hall, 1989, 412 p.

25. **Patterson D., Hennessy J. L.** *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*, Morgan Kaufmann, 2017, 696 p.

26. **Nikolaev D. S., Korneev V. V.** The use of container virtualization mechanisms in high-performance computing complexes with the SLURM task scheduling system, *Programmnaya Ingeneria*, 2017, vol. 8, no. 4, pp. 147—160. DOI: 10.17587/prin.8.147-160 (in Russian).

ИНФОРМАЦИЯ

DOI: 10.17587/prin.13.483-496

Reanimator: from Test Data to Code and Back

A. M. Abdullin, Post-Graduate Student, Senior Lecturer, azat.aam@gmail.com,
V. M. Itsykson, PhD, Associate Professor, vlad@icc.spbstu.ru,
Peter the Great St. Petersburg Polytechnic University (SPbPU), Saint-Petersburg, 195251, Russian Federation

Corresponding author:

Azat M. Abdullin, Post-Graduate Student, Senior Lecturer,
Peter the Great St.Petersburg Polytechnic University (SPbPU), Saint-Petersburg, 195251, Russian Federation
E-mail: azat.aam@gmail.com

Received on September 09, 2022
Accepted on October 10, 2022

State-of-the-art automatic testing tools can efficiently detect various kinds of errors in software projects; however, these errors often cannot be automatically extracted to a standalone reproducing test case, a feature valuable for the purposes of further software maintenance. Objects present the main difficulty, as generating code to construct an object given its state is hard, due to the encapsulation principle.

In this paper we present an approach called Reanimator that, given an object representation, is able to generate a valid code snippet which constructs this object using its publicly available API. Reanimator can be applied to any automatic testing tool to help it generate reproducing test cases for detected failures.

We implemented our approach as a part of an automatic testing tool called Kex and evaluated it on a number of open-source projects from GitHub. Evaluation results showed that our approach is compatible with the state of the art techniques.

Keywords: *automatic test generation, symbolic execution, software testing*

For citation:

Abdullin A. M., Itsykson V. M. Reanimator: from Test Data to Code and Back, *Programmnaya Ingeneria,* 2022, vol. 13, no. 10, pp. 483—496.

УДК 004.05

А. М. Абдуллин, аспирант, ст. преподаватель, azat.aam@gmail.com,
В. М. Ицыксон, канд. техн. наук, доц., vlad@icc.spbstu.ru,
Санкт-Петербургский политехнический университет Петра Великого

Реаниматор: от тестовых данных к коду и обратно

Современные инструментальные средства для автоматической генерации тестов способны эффективно находить различные виды ошибок в программном обеспечении. Однако эти ошибки зачастую не могут быть автоматически преобразованы в воспроизводимый тестовый пример. Такое преобразование позволило бы значительно упростить процесс поддержки разрабатываемого программного обеспечения. Наибольшее сложным вопросом решения этой задачи является восстановление экземпляров классов, так как в силу принципа инкапсуляции очень сложно сгенерировать программный код, который позволяет воспроизвести необходимое состояние объекта.

В статье описан оригинальный, именуемый Реаниматором, авторский подход, который, получив описание необходимого состояния экземпляра класса в каком-либо формате, формирует программный код, результатом которого является создание необходимого экземпляра в нужном состоянии с использованием исключительно публичного интерфейса класса. Такой подход может быть использован для создания тестов в любом инструментарии для автома-

тической генерации тестов, воспроизводящих найденные ошибки. Предлагаемый подход был разработан и реализован в составе инструментального средства Kex для автоматической генерации тестов и протестирован на наборе проектов с открытым исходным кодом. Тестирование показало применимость и конкурентоспособность предлагаемого подхода в сравнении с современными методами генерации тестовых примеров.

Ключевые слова: *автоматическая генерация тестов, символьное исполнение, тестирование программного обеспечения*

Introduction

Software affects almost every part of human life in the modern world. People are surrounded by computers and electronic devices which control transport, business, medicine, etc. Software defects in these computer systems may lead to fatal consequences, and software companies use various methods of software quality assurance [1, 2] to detect and fix errors on all stages of development.

One of the most popular methods of software quality assurance is software testing. Testing involves executing the program (or one of its components) on a set of predefined inputs to detect incorrect behavior. Software testing has proved its efficiency, but it has one significant weakness: creating software tests is a very tedious process requiring a lot of time-consuming manual work [3].

Automatic test generation is one of the attempts to overcome this weakness; being an extension of automatic software testing, it not only finds the software bugs, but also generates reproducing test cases, usable for further software maintenance. Generated tests are often included in the regression test suite to ensure that the found problems will not reappear in the future. There are a lot of methods of automatic test generation, based on random testing [4], symbolic execution [5], hybrid search-based approaches [6], etc. These methods may use different strategies for test generation, but in the end they all produce error-inducing inputs by either operating on *test code* or on *test data*.

Methods from the first group [6—8], which primarily use flavors of random search (e. g., evolutionary algorithms), operate on code in the form of function or method calls and work by extending the generated call sequence with additional calls on each search step. This allows these methods to produce a code snippet which can be immediately used in the testing environment; however, their random nature has a direct impact on their bug-revealing efficiency. Another downside of these approaches is that they are dynamic by nature and cannot be used in purely static setup.

Methods from the second group [5, 9, 10], based on random data generation or more complex approaches, such as symbolic execution, operate on data rather than code and try to generate interesting program inputs. Therefore, to turn the error-inducing input into a test case, one needs to generate a code snippet which constructs the required data. For programming languages following encapsulation principle (e. g., object-oriented ones), this leads to the following problem: one cannot directly build an object from its internal data (aka "encapsulation problem"). This forces the test generation tools to use alternative ways to create test cases, such as reflection [11].

Post-failure debugging techniques analyze the program data after the failure and try to lift the data into a test case. Given a bug report in some form (e. g., core dump, stack trace, etc.), they attempt to generate program inputs (execution traces, test cases, etc.) which reproduce the crash. However, they are also affected by the "encapsulation problem", if the program input is complex.

Record and replay [12, 13] crash replication approaches fall somewhere in between these two groups. These methods capture program interactions at runtime (via instrumentation) and store them as execution traces. Test cases can be later recreated from these traces, based on the sequences of program events, and these sequences correspond to test code fragments. However, one may need to generate the test data for these fragments, which brings us back to the "encapsulation problem". Additionally, the need for instrumentation often leads to a performance overhead.

This paper presents an approach we call Reanimator which solves the "encapsulation problem" and allows one to automatically generate valid code snippets to create a given target object, using its *publicly available API*; the generated snippets can be then used to build reproducing test cases. The approach is general and can be used not only to solve the "encapsulation problem" for data-based test generation methods, but also to enhance code-based test generation, e. g., by using the object creation snippets as primitive elements during the call sequence search process.

Reanimator is built on an original backward search algorithm, which works by gradually reducing the search space of applicable actions initializing one or more object fields. At each step of the search algorithm, we take the target object T and construct another source object S, so that all fields of S are equal to corresponding fields of T except for one, set to the default value of its type. We then use symbolic execution to check if it is possible to execute an action on S to produce the target object T. If it is possible, we save the information about found action

and its arguments, and recursively continue the search on *S*, until we find a constructor-like action, meaning we can create *T* without any source object. The resulting action sequence can be later transformed into a valid test case.

We implemented our approach as a prototype in an automatic testing tool provided in Kex [14] platform and evaluated it on a number of real-world projects. The evaluation consists of three scenarios: extracting errors found by the white-box fuzzer component of Kex to the reproducing test cases, creating test cases for generating random objects and integrating Reanimator approach with the TARDIS [15] tool. Evaluation results show Reanimator can successfully and efficiently generate 62.3 % of target objects on average and it can be compatible with the dynamic approach of TARDIS. Based on the evaluation we can say the proposed approach is applicable for automatic test case generation.

1. Related work

Automatic generation of reproducing code snippets (i. e., test cases) from error-inducing inputs is a problem that has been addressed in areas of automatic test generation, debugging and crash reproduction. Different approaches have different capabilities for generating test cases, here we talk about the most important ones.

Some of the approaches [6—8, 16] are based on the idea of constructing the test case by gradually adding new code expressions from the list of available expressions until the user-specified stop criterion is reached (not unlike a form of program synthesis [17]). While these approaches are great at generating test cases (which they perform by construction), the generation process is usually random to some extent and may not be able to create test cases that cover complex parts of the target program. Also, those approaches are based on dynamic analysis of the program, as they are usually coverage driven.

Symbolic execution based approaches differ on how exactly they use symbolic execution for test case generation. Some of them (i. e., JBSE [5]) find interesting object states via symbolic execution and generate test cases using reflection to reconstruct the target objects. While reflection does sidestep the "encapsulation problem", it has the following downsides:

- reflection-based tests are hard to comprehend and maintain;
- using reflection breaks encapsulation and allows generating an object that cannot be created via its publicly available API.

Other symbolic execution approaches (e. g., Symstra [9]) are exploring the test case search space for an arbitrarily selected subset of possible call sequences. During the exploration, they symbolically execute a call sequence and, if the execution is interesting, generate concrete values for the arguments. However, Symstra is limited in that it only supports primitive types; the authors argue constructing complex objects can be done either using reflection or by viewing it as a nested Symstra problem, which may increase the search space.

JBSE [5] symbolic execution engine is also used as a basis for SUSHI [10] and TARDIS [15] test generation tools. SUSHI uses JBSE to compute interesting path conditions of target program and converts them into a fitness function. This fitness function is then used in the search algorithm which generates test cases satisfying the fitness function and, therefore, the original path conditions; to implement search, SUSHI uses a custom version of EvoSuite [6]. The authors also propose a way to encode the program input properties as external specifications, which can help SUSHI to identify the unsatisfiable path conditions in advance and improve its efficiency.

TARDIS [15] is an extension of SUSHI that uses concolic testing [18] for exploring path conditions of the target program, which are then also handled via a fitness function. The use of concolic testing improves TARDIS performance (compared to SUSHI) in cases when there is no user provided specification. The main disadvantage of these test generation approaches is that the search algorithm is separated from the symbolic execution engine, making it extremely hard to efficiently evaluate the feasibility of interesting path conditions. The results presented in [10, 15] and our experiments show that most of the time is spent on search-based test case generation, compared to time taken by the symbolic execution used to explore interesting path conditions.

Crash replication is another area interested in test case generation. Approaches in this field can be divided into two main categories: record and replay approaches and post-failure techniques. The former (SCAPE [12], CR [19], JINSI [20], BugRedux [21], ReMinds [22], GenThat [13]) are all based on capturing program execution traces via instrumentation and replaying these traces later, either in a special execution environment or as generated unit tests. However, the unit tests created do not support complex data or utilize reflection; more so, program instrumentation can lead to additional performance overhead.

Post-failure techniques [23—27] analyze execution data (e. g., core dumps) after the program has already crashed. RECORE [24] uses core dumps to generate a fitness function and provides it to evolutionary search-based algorithm to generate the test cases, somewhat similar to SUSHI (and having the same disadvantages). SBFR [27] (search-based failure reproduction) takes a failing program, a grammar describing the complete program input, and a (partial) call sequence and uses genetic algorithm to generate failing inputs. While such approach works for functional tests, the need to create an input grammar for separate unit tests adds a significant overhead.

ESD [23] is a technique for automated debugging which, given a program and a bug report, uses symbolic execution to synthesize a program execution which reproduces the required bug. The execution can then be replayed in a special ESD playback environment to support classic debugging techniques, but it does not provide a way to generate a reproducing test case. DESCRY [25] is an automated tool for reproducing system-level concurrency failures based on log messages collected from the running program. Once again, it is aimed at functional-level analysis and does not provide utilities to generate separate unit tests.

Summary. As one can see, there already exists a large body of work on automatic generation of test cases. Existing methods belong to one of the following high-level groups:

- complete test code generation via random search;
- end-to-end test data generation, which primarily supports functional-level tests with primitive data types;
- unit-level test data generation, which comes in two flavors:
  - using reflection to create the required data;
  - using search-based approaches to synthesize code creating the required data.

For unit tests, reflection-based approaches sidestep the "encapsulation problem", but create hard to maintain and even incorrect (with reference to program API) tests. Search-based approaches do not have this problem, but sacrifice performance with the separation between symbolic execution (used for data exploration) and code synthesis (used for data generation).

Limitations of both approaches have inspired us to attempt to remove the separation and use symbolic execution for both data exploration and generation.

2. Motivating example

Let us show a motivating example of how current test generation tools support complex data generation. Consider a simple Java class ListExample given in Listing 1. It defines an inner class Point with two integer coordinates and defines a method foo which accepts ArrayList<Point> as its argument. This method fails on line 18, when argument *a* has a specific shape.

Imagine we want to generate a test case for this example. Automatic test generation tools based on random- and search-based code generation will have difficulties finding this failure, as the probability they will synthesize code, which creates an object of required shape to trigger the bug, is low. Tools using symbolic execution, on the other hand, will be able to find the error-inducing input, but will encounter some problems generating a standalone test case. As discussed in the previous section, one of the options is to use reflection. Listing 2 shows a part of the test suite for our example program generated by JBSE [5] tool. This example

```
1. package test;
2. import java.util.ArrayList;
3.
4. public class ListExample {
5.     public static class Point {
6.         private int x;
7.         private int y;
8.         public Point(int x, int y) {
9.             this.x = x; this.y = y;
10.        }
11.        public int getX() { return x; }
12.        public int getY() { return y; }
13.    }
14.    public void foo(ArrayList<Point> a) {
15.        if (a.size() == 2) {
16.            if (a.get(0).getX() == 10) {
17.                if (a.get(1).getY() == 11) {
18.                    throw new IllegalStateException();
19.                }
20.            }
21.        }
22.    }
23.}
```

Listing 1. A program with a hard-to-find bug

```
public class TestSuite {
    ...
    public void test4() {
        this.nullObjectFields = new HashSet<>();
        ...
        test.ListExample __ROOT_this = (test.ListExample)
            newInstance("test.ListExample");
        java.util.ArrayList __ROOT_a = (java.util.ArrayList)
            newInstance("java.util.ArrayList");
        new AccessibleObject(__ROOT_a)
            .set("java/util/ArrayList:size", 2L);
        new AccessibleObject(__ROOT_a)
            .set("java/util/ArrayList:elementData",
                newArray("java.lang.Object", 2L));
        new AccessibleObject(__ROOT_a)
            .set("java/util/ArrayList:elementData[0]", null);
        this.nullObjectFields.add(new ObjectField(
            __ROOT_a, "java/util/ArrayList:elementData[0]"));
        __ROOT_this.foo(__ROOT_a);
    }
    ...
}
```

Listing 2. Example of a test case generated by JBSE

highlights the problems with the use of reflection we mentioned earlier. The total size of a single generated test is more than 400 lines, and the generated code is complex and hard to understand and to maintain.

Tools, which use search-based approaches to create the test code for interesting data (such as SUSHI or TARDIS), encounter performance problems. Our experiments show SUSHI is not able to generate a test case triggering the bug in line 18 with a time budget of 20 min. TARDIS is more successful in this case and can generate a test case in 2 min (Listing 3). These tools require more time for test case generation; however, the created tests do not break encapsulation and are easier to comprehend and maintain if needed.

```
public class ListExample_0_Test extends ListExample_0_Test_scaffolding {
  @Test(timeout = 4000)
  public void test0() throws Throwable {
    ListExample listExample0 = new ListExample();
    ArrayList<Point> arrayList0 = new ArrayList<Point>();
    int int0 = 10;
    Point point0 = new Point(int0, int0);
    int int1 = 11;
    Point point1 = new Point(int1, int1);
    boolean boolean0 = arrayList0.add(point0);
    boolean boolean1 = arrayList0.add(point1);
    try {
      listExample0.foo(arrayList0);
    } catch (IllegalStateException e) {
      verifyException("org.example.ListExample", e);
    }
  }
}
```

Listing 3. Example of a test case generated by TARDIS

Reanimator attempts to combine the best of both worlds, by supporting direct generation of test code from interesting data without the need to use reflection.

3. Reanimator

Given a target object representation (as an error-inducing input from the testing tool), Reanimator generates a code snippet for creating the object. The approach was originally developed for the JVM platform; thus, its description contains several JVM-specific features, but it can be adapted for most general-purpose programming languages. We assume a *closed-world model*, i. e., we have full access to all types, functions, etc. Reanimator can be divided into three stages:

1) *descriptor* conversion: descriptors are the internal object representation used in Reanimator;

2) *action sequence* generation: each action is an operation (e. g., function call or array element access) in the target language;

3) *code snippet* generation: code snippet is a valid code sample which creates the target object.

3.1. Descriptor conversion

First, the target object should be converted to a Reanimator descriptor. Descriptors are used to represent the object shape; one may consider them to be trees which capture (nested) object states. The descriptor format for the JVM platform is given in Listing 4; if needed, the format can be extended to support other languages. To convert an object representation to a descriptor, one may follow its shape in a bottom-up fashion, converting object elements to their corresponding descriptors along the way.

3.2. Action sequence generation

At this stage Reanimator tries to create a sequence of valid actions which, if performed, create an object corresponding to the target descriptor. The list of supported actions in the current implementation for the JVM platform is as follows:

- constructor-like calls:
  - no-arg constructor calls;
  - constructor call with arguments;
  - external constructor-like call (static factory methods, etc.);
- (static) method calls;
- (static) field assignments;
- array creations;
- array element writes;
- primitive value creations;
- "unknown" actions.

```
<Descriptor> ::= "ConstantDescriptor"
     "ObjectDescriptor" fields:<ListOfFields>
     "ArrayDescriptor" elements:<ListOfElements>
     "StaticFieldDescriptor" field:<Field>

<ConstantDescriptor> ::= "NullDescriptor"
     "BoolDescriptor" value:Boolean
     "ByteDescriptor" value:Byte
     "ShortDescriptor" value:Short
     "CharDescriptor" value:Char
     "IntDescriptor" value:Int
     "LongDescriptor" value:Long
     "FloatDescriptor" value:Float
     "DoubleDescriptor" value:Double

<Field> ::= name:String klass:Class value:<Descriptor>

<Element> ::= index:Int value:<Descriptor>

<ListOfFields> ::= <Field> <ListOfFields> | <empty>

<ListOfElements> ::= <Element> <ListOfElements> | <empty>
```

Listing 4. JVM descriptor format

```
Input: d — target descriptor
Input: limit — action sequence length limit
Output: calls — generated action sequence
 1: function generate(d, limit)
 2:     if 0 == limit then
 3:         return unknown
 4:     end if
 5:     calls ← []
 6:     if d ∈ ConstantDescriptor then
 7:         calls += PrimitiveValue(d.value)
 8:     else if d ∈ StaticFieldDescriptor then
 9:         value ← generate(d.value, limit − 1)
10:         calls += StaticFieldSetter(d, value)
11:     else if d ∈ ArrayDescriptor then
12:         eType ← d.elementType
13:         length ← generate(d.length, limit − 1)
14:         arr ← NewArray(eType, length)
15:         calls += arr
16:         for (idx, ed) ∈ d.elements do
17:             value ← generate(ed, limit − 1)
18:             calls += ArrayWrite(arr, idx, value)
19:         end for
20:     else if d ∈ ObjectDescriptor then
21:         calls += generateObject(d, limit − 1)
22:     end if
23:     return calls
24: end function
```

Figure 1. Action sequence generation algorithm

Reanimator respects the encapsulation principle and uses only publicly available program actions, e. g., field assignments are allowed only for public fields. A high-level overview of action sequence generation is shown in Figure 1. Generation of action sequences for constant descriptors, array descriptors and static field descriptors is straightforward and self-explanatory. Generation of object descriptors, however, is more complex; let us discuss it in more detail.

As shown in Listing 4, object descriptor is represented as a list of fields with their types and values. If a field is not important in the context of current test generation (i. e., it is irrelevant with reference to error-inducing input), it is not included in the object descriptor.

Each field of the object descriptor imposes new constraints for the object generation. Generation of an object descriptor with n defined fields is strictly more complex than generation of another object descriptor with $m < n$ defined fields. This reduction-like intuition is the basis of the action sequence generation algorithm for object descriptors presented in Figure 2.

The algorithm gradually reduces the descriptor until it finds a constructor-like call, which can directly create the object. A naive approach to doing this is to check all possible valid action sequences, but such brute-force search is very inefficient and may not terminate in a reasonable time in some cases. To overcome this problem, we apply symbolic execution, using it to offer several optimizations to speed up the search of interesting object actions, and impose a hard limit to ensure generation termination.

As the first step, we perform descriptor concretization. The purpose of this is to replace all non-instantiable types in object descriptors with arbitrary instantiable ones. Currently, a type is considered non-instantiable if it is an abstract class or an interface, as they are not constructible directly. Finding compatible types can be done efficiently, as Reanimator operates under the closed-world assumption.

The next step is setter extraction, which is optional, and its main purpose is to speed up the search. We consider method a "setter", if it takes exactly one argument and changes exactly one of the object fields. The main idea is to preprocess available classes and find their setters (if present). Then, if an object descriptor contains fields with available setters, we shortcut and generate corresponding setter call actions, add them to the sequence and reduce these fields from the target descriptor. The details of how we check if a method is a setter are covered in more detail in the following section.

```
Input: d — target object descriptor
Input: limit — action sequence length limit
Output: calls — generated action sequence
 1: function generateObject(d, limit)
 2:     d ← concretize(d)
 3:     ctors ← d.class.ctorLikeCalls
 4:     methods ← d.class.methods
 5:     (d, setters) ← extractSetters(d)
 6:     query ← {d, setters}
 7:     calls ← []
 8:     while query ≠ ∅ do
 9:         if length(calls) > limit then
10:             return unknown
11:         end if
12:         (desc, calls) ← query.poll()
13:         for ctor ∈ ctors do
14:             (nDesc, args) ← execAsCtor(ctor, desc)
15:             if isFinal(nDesc) then
16:                 margs ← genArgs(args, limit)
17:                 calls += CtorCall(ctor, margs)
18:                 return calls
19:             end if
20:         end for
21:         for m ∈ interestingFor(desc, methods) do
22:             (nDesc, args) ← execAsMethod(m, desc)
23:             if nDesc ⩽ desc then
24:                 margs ← genArgs(args, limit)
25:                 nCalls ← calls + MethodCall(m, margs)
26:                 entry ← {nDesc, nCalls}
27:                 query.push(entry)
28:             end if
29:         end for
30:     end while
31: end function
```

Figure 2. Object descriptor processing

After that, the main search procedure is started. At each step, we first check for the termination condition; if we stop the search prematurely, we signal this with a special "unknown" action. Then we check if it is possible to create the given descriptor using any constructor-like call; if that is true, the action sequence generation is complete. We create a constructor call action, add it to the sequence and return the result.

If no constructor-like calls are applicable, the search continues to iterate over all interesting methods, attempting to reduce the descriptor. A method is interesting with reference to given descriptor, if it is public (i. e., accessible from tests) and changes at least one of the descriptor fields. If we were able to successfully reduce the descriptor using one of the methods, we generate arguments for that method with *genArgs* function, which recursively calls *generate* for all argument descriptors. Then we build a method call action, add it to the current sequence and schedule it for processing.

We should also note that *generate* and *generateObject* also save all information about generated descriptors to cache allowing them to support generation of cyclic descriptors.

Functions *execAsCtor* and *execAsMethod* are used to symbolically execute a callable (a constructor or a method) to check if it can be used to create or reduce the target descriptor. If the check is successful, they return the reduced descriptor and arguments descriptors needed to successfully call it. Internally, these functions use SMT solvers [28] to reason about the behavior of callables. Let us describe how these functions work in more detail.

3.2.1. Reducing descriptors using symbolic execution

Applying symbolic execution to reason about callables and their influence on object state is what allows Reanimator to efficiently reduce descriptors during action sequence generation, while respecting encapsula-

tion principle. Our instance of SMT based symbolic execution consists of the following steps:

- encode the callable and the descriptor as SMT formulae;
- perform an SMT query into the SMT solver;
- decode a new descriptor from the resulting SMT model.

To be able to use SMT solver for symbolic execution, we need to define a memory model suitable for representing the program and its variables as SMT formulae. The memory model used in Reanimator is inspired by the work on Kex automatic test generation tool [14].

JVM bytecode has several primitive data types: booleans, integers (short, int, etc.), *floating* point numbers (*float and double*). Each variable of a given type can be represented as an expression of corresponding SMT theory: *booleans* for boolean, *bitvectors* [29] for integers, *floating point numbers* [30] for float and double.

JVM also supports non-primitive data types in the form of reference types (objects and arrays). To represent references in the heap we use a "property-based" memory model [31]: memory is encoded as a collection of SMT arrays [32], each array corresponding to a disjoint partition of heap objects not aliasing object from other partitions. This allows to encode object references as 32-bit bitvector indices into their partition; arrays are represented as continuous chunks, with array reference pointing to its start index. Object fields are represented in a similar fashion, using "property memories": each field is mapped to a separate SMT array, indexed by object references; to access field *x.y* one needs to work with property memory *typeOf(x).y* by index *x*. This allows for precise modeling of heap structures while also reducing the complexity of solving the resulting formulae, as disjoint SMT arrays decrease the search space SMT solver needs to work with.

Runtime type information is encoded in a special "type" property memory: each reference may be used as an index to this property memory to get its type. As we analyze the program as a closed-world system, we can assign a constant to each type and encode subtyping via SMT axioms over *isSubtype* uninterpreted function, which encodes all the available type information.

All type-related operations in the program are expressed through *isSubtype*: casts and *instanceof* checks impose new constraints on the "type" property of the corresponding variable. That, together with the subtyping axioms, gives SMT solver enough information to correctly analyze types.

3.2.2. Encoding the SMT formulae

Methods are encoded as SMT formulae, using the described memory model. Encoding most JVM instructions is straightforward, as they can be directly mapped to corresponding SMT expressions (*iadd* to *bvadd*, *aaload* to *select*, etc.). However, there are some typical problems: loops and method calls cannot be encoded as SMT formula as easily.

To overcome these problems, we use the following processing. All loops are unrolled to a predefined bound, underapproximating the possible method behavior, similarly to how bounded model checking works [33]. This unrolling allows the methods to be converted into an SMT formula. Function calls are inlined if they can be statically resolved, otherwise, they are underapproximated as "noop" operations. For test case generation, our experiments show such underapproximations are good enough for most practical cases.

3.2.3. Performing an SMT query

After the methods are represented as SMT formulae, which symbolically capture their behavior, they are used together with the descriptors to answer different queries with reference to the action sequence generation algorithm. The purpose of these queries is to understand how a given method affects an object's state, i. e., how the final object state is changed in comparison with its initial state after the method invocation.

SMT queries to answer that are constructed as follows. The target descriptor defines the constraints for the final memory state: properties, corresponding to the defined descriptor fields, should be assigned their value in the final state. Method type defines the constraints for the initial memory state:

- constructor constraints, used to check if the target descriptor can be created by this constructor; they require the complete initial memory state to be uninitialized;
- setter constraints, used to check if a method can be used to set one or more target fields to their descriptor values; they require all target field values in the initial memory state to be uninitialized;
- method constraints, used to check how method execution affects the object state; they do not impose any constraints on the initial memory state.

Function *execAsCtor* is checking queries with constructor constraints, function *execAsMethod* — queries with setter and method constraints. Setter constraints are also used in the setter extraction.

We need to analyze method constraints as well as setter constraints, because some fields of an object may not have a direct setter but could be modified by other methods. For example, the size field of a collection cannot be directly manipulated, but it is changed after calls add or clear.

3.2.4. Decoding new descriptors

A successful SMT query returns an SMT model containing the initial memory state, the final memory state, and values of all method variables. To extract the

```
package test;
import java.lang.Throwable;
import java.lang.IllegalStateException;
import org.junit.Test;
import test.ListExample;
import test.ListExample.Point;
import java.util.ArrayList;

public class ListExample_foo {
  public <T> T unknown() {
    throw new IllegalStateException();
  }
  @Test
  public  void test_bb10() throws Throwable {
    ListExample listExample1 = new ListExample();
    ArrayList<ListExample.Point> al =
        new ArrayList<ListExample.Point>(2);
    ListExample.Point point1 = new ListExample.Point(10, 0);
    al.add(point1);
    ListExample.Point point2 = new ListExample.Point(0, 11);
    al.add(point2);
    listExample1.test(al);
  }
}
```

Listing 5. JVM descriptor format

information, we need to decode this model into new, reduced descriptors.

The decoding process consists of evaluating the values of all interesting fields from the initial memory state, to understand which constructor, setter or method constraints were not violated. Fields that have the uninitialized value in the initial memory state are considered successfully reduced and are not included in the resulting descriptor. If the constructor constraints were not violated (the complete initial memory state is uninitialized), the descriptor is considered successfully generated.

3.3. Code snippet generation

After we have created an action sequence for the target object, we need to convert it into a test case code snippet. As the action sequence is a list of callable actions with their arguments, which create the needed object, the transformation is straightforward.

To get a complete code snippet, we need additional information, specifically, resolved type and type parameter information, if we want to correctly use the public API and avoid using reflection. The required type information can be extracted by traversing the action sequence two times: in backward and forward direction. In the prototype implementation, backward traversal uses Java reflection to obtain type parameter types. Forward traversal uses the results of the backward one and resolves the final types for action calls and variables, which are used as follows:

- parameter types can provide type arguments for executable calls;
- explicit type casts are added if an argument variable has type incompatible with the parameter type.

The rest of code snippet generation involves mechanical extraction of sequence actions into code. The extracted Java snippet for our example is shown in Listing 5. The *unknown* function is generated to support the special unknown action, meaning Reanimator can create a test case even if the generation failed, supporting the option of manual developer intervention.

4. Implementation

We implemented the Reanimator approach in an automatic testing tool provided in Kex [14] platform. Kex is a white-box [34] testing tool targeting JVM bytecode, and uses Kfg library[1] to analyze .jar files and construct control flow graphs (CFG). Kfg is also used to perform various bytecode transformations and simplifications, e. g., loop unrolling's.

Kex works as a symbolic execution engine and uses SMT solvers to perform constraint solving. Instead of working directly with Kfg CFGs, Kex uses its own

[1] https://github.com/vorpal-research/kfg

intermediate representation called *predicate state*. Predicate state serves as an inter-layer between Kfg and SMT formulae, allowing it to easily support multiple solvers (Boolector [35], Z3 [36], STP [37]). It is also used to perform additional, SMT-specific transformations on the program code.

5. Evaluation

To evaluate Reanimator, we ran the prototype implementation on a set of open-source projects. First, we selected projects from JUnitContest 2020 [38] benchmark set: fescar-0.1.0, pdfbox-2.0.18 and spoon-7.2.0. Second, we selected several open-source projects from GitHub: authforce-core-13.3.0, exp4j-0.4.9, exposed-0.27.1, imixs-workflow-4.4.6, karg-0.1, koin-2.1.6, kotlinpoet-1.7.0, kfg-0.0.10. We tried to diversify the test projects by picking them from different application domains, such as command line parsing, SQL libraries, code generation utilities, etc.

All tests were run on a test system with 64-bit Arch Linux OS (kernel 5.8.13-arch1), Intel Core i7-4790 CPU @ 3.60GHz, 32GB of RAM and Samsung SSD 950 PRO 512GB storage. We used the following Reanimator configuration:

- Z3 solver for SMT solving;
- all loops are unrolled to bound $k\_l = 2$;
- the length of action sequences set to $limit = 5$.

We have chosen values for parameters k_l and *limit* based on the preliminary experiments with the following reasoning. As we discussed in Section 3, unrolling is used for underapproximating the method behavior, with larger unroll bound making it more precise, but also more difficult to solve. Our experiments showed larger unroll bounds do not significantly improve the generation quality; therefore, we selected k_l to be relatively small. The length of action sequences *limit* is directly affecting both Reanimator performance and the final generation quality. However, while increasing *limit* increases Reanimator generation time, the improvement in generation quality quickly decreases. We selected *limit*, so that the Reanimator can finish code snippet generation in reasonable time. However, one may select other values for k_l and *limit* if needed; it should not have any impact on the applicability of Reanimator, and influence only its performance.

In the evaluation, we address the following research questions (RQ).

RQ1. Can Reanimator be used in automatic test generation tool to create valid test cases? We want to understand what percentage of interesting inputs generated by Kex can be converted to valid code snippets using Reanimator.

RQ2. Can Reanimator be used to generate valid code snippets for random target objects? We want to see if our approach can create code snippets for randomly generated objects and analyze what causes it to fail.

RQ3. How does Reanimator compare to other existing approaches? We want to know how Reanimator performs in comparison with TARDIS automatic test generation tool.

5.1. RQ1: Using Reanimator as a test generator for Kex

We address RQ1 by using our prototype as a "backend" for test generation from the output of Kex. In its default mode, Kex uses Java reflection library to create objects from the symbolic output of an SMT solver (similarly to JBSE). In this part of the evaluation, instead of reflection we used the Reanimator to generate code snippets. Being a white-box tool, Kex tries to cover each basic block of each method by generating interesting input data. The achieved coverage is measured as instruction coverage.

The results of the evaluation are presented in Table 1. The first column contains the name of the tested project. The second column shows project line coverage, as a relative measure of the project complexity. The third column contains the percentage of successfully generated code snippets for all interesting inputs found by Kex (**AG**). The fourth column shows the average depth of descriptors (depth of a descriptor is the maximal nesting level of its elements) (**ADD**). As we found during evaluation, many descriptors generated by Kex are simple constant descriptors or empty object descriptors, for that reason we decided to additionally measure non-trivial descriptor parameters. The fifth column shows

Table 1

Results of using Reanimator as a part of Kex

Project	Coverage, %	AG, %	ADD	NTG, %	NTDD
authforce	13.9	71.8	1.5	48.5	2.5
exp4j	27.5	79.7	1.6	4.7	2.8
exposed	29.0	70.0	1.3	45.3	2.3
fescar	21.7	86.1	1.4	47.3	2.6
imixs	25.1	89.3	1.5	70.1	2.3
karg	15.8	66.0	1.5	38.6	2.3
kfg	21.7	58.1	1.4	42.0	2.4
koin	33.2	70.4	1.9	59.3	3.0
kotlinpoet	28.5	76.3	1.6	65.8	2.6
pdfbox	9.8	83.4	1.6	75.1	2.3
spoon	5.9	83.8	1.3	40.6	2.3
average	21.1	75.9	1.5	48.8	2.5

the percentage of successfully generated code snippets for non-trivial descriptors (**NTG**). The last column shows the average depth of non-trivial descriptors (**NTDD**).

The results show that on average Reanimator can successfully generate code snippets in 48.8 % of non-trivial cases and in 75.9 % of all cases. The average depth of non-trivial descriptors over all projects is 2.5. Manual analysis of the results has showed the main reasons for Reanimator failures are code complexity (w.r.t. our underapproximation), "impossible" objects (objects that cannot be created using the public API) and higher-order functions.

The results also show that Reanimator performed poorly on exp4j project. That happened because exp4j project contains a lot of package-private and anonymous classes which Reanimator cannot generate.

Based on that, we believe the proposed approach can be used in automatic test generation tools to create code snippets from input data.

5.2. RQ2: Generating code snippets for creating random objects

To address RQ2 we evaluated our prototype on generating random objects for classes from the test projects. We used the following evaluation technique. For each test project we generated a set of random objects using easy-random-4.2.0 library[1]. From this set we have selected a subset of "valid" objects, i. e. objects with public visibility and at least one public constructor-like callable. We then used our prototype to try and generate code snippets for valid objects, until we got 1000 successful generations for each project. To estimate the complexity of random objects we also measured the average depths of generated descriptors.

To check the correctness of the generation, we need to have an oracle: a way to ensure the generated object is structurally equal to the target one. We cannot use equals implementations because they are not required to test structural equality. To sidestep that, we implemented an external structural equality test.

- For a target object *a*, we convert it into a descriptor *a_d*.
- We generate an action sequence for *a_d* and use it to create a generated object *b*.
- The generated object *b* is converted into a descriptor *b_d*.
- We check the descriptors for equality: $a\_d = b\_d$.

By construction, if the generated object is structurally equal to the target object, their corresponding descriptors should be equal.

The results of random object generation are presented in Table 2. The first column shows the name of the project. The second and third columns show the sucess rate for generation of valid objects (**AG**) and average depth of the descriptors for valid objects (**ADD**). During the experiments we noticed the Reanimator cannot successfully generate unordered collections (sets and maps) in most cases, as their complex internal structure presents problems for the symbolic execution. For that reason, we decided to also measure success rate (column **VG**) and average depth (column **VDD**) for *viable descriptors*, i. e. descriptors without complex collections.

Table 2

Results of generating random objects

Project	AG, %	ADD	VG, %	VDD
authforce	32.4	3.2	40.4	3.0
exp4j	41.1	2.6	47.9	2.3
exposed	42.4	3.3	49.9	2.9
fescar	46.6	3.1	63.5	2.1
imixs	67.4	3.5	80.4	2.8
karg	75.5	3.4	75.5	3.4
kfg	66.3	3.8	95.3	3.1
koin	38.7	3.8	50.1	3.1
kotlinpoet	29.7	2.5	63.7	1.5
pdfbox	28.5	5.2	38.5	4.0
spoon	37.0	3.6	44.1	3.3
average	45.9	3.5	59.0	2.9

We can see that the average success rate is 45.9 % for valid objects and 59 % for viable descriptors; overall, the success rate is very dependent on the target project. For example, pdfbox project has a lot of classes that operate with complex PDF document structures. Reanimator is not always able to resolve these complex structures, therefore, the success rate is relatively low. On the other hand, project kfg has a lot of data classes: immutable classes with public "setter" constructors, allowing to initialize all object fields at once, thus, the success rate for kfg is high.

The amount of generated objects does not allow us to manually inspect each Reanimator failure and the complexity of such task does not allow performing it automatically. Therefore, we inspected a subset of 120 Reanimator failures to analyze their causes. 35 % of the failures were caused by impossible objects generated by easy-random. The main reason for other failures is the complexity of the generated objects (58 %). Among other reasons for the failure are the use of higher-order functions (4 %) and "builder" pattern (3 %), which are currently not supported by Reanimator.

[1] https://github.com/j-easy/easy-random

The last group of failures can be explained by the prototype implementation limitations. The approach can be easily extended to support the builder pattern. As for the higher order functions, it should be possible to search for existing functions with the required signature or even generate them on-the-fly. Another way to improve Reanimator success rate is to improve the symbolic execution by adding support of unordered collections or improving its underapproximation. We consider these tasks as future work.

5.3. RQ3: Comparing Reanimator to TARDIS approach

To address RQ3 we have integrated Reanimator to the TARDIS tool and compared it to the default approach used in TARDIS on the JUnitContest 2021 benchmark set. JUnitContest 2021 used the time budgets of 60 and 120 s, but we decided to also run TARDIS on the increased time budgets as the authors of TARDIS suggested that 120 s is too low. We have also evaluated Kex with Reanimator on the same benchmark set.

The results of the evaluation are presented in Table 3. The first column shows the name of the project and the approach used for code snippet generation. Other columns show average line coverage achieved by the tools on the benchmark classes. We can see that TARDIS with Reanimator performed similarly as TARDIS with Evosuite. Thus, we can say that Reanimator is an applicable approach for automatic test case generation tools.

Table 3

Results on JUnitContest 2021 benchmark, %

Tool	Time budget, s			
	60	120	300	600
TARDIS + Evosuite	14.0	15.7	18.5	19.6
TARDIS + Reanimator	13.9	16.0	17.8	19.3
Kex + Reanimator	24.6	25.3	25.4	27.6

Conclusion and future work

In this paper we presented an approach called Reanimator for generating valid code snippets to create a given target object, using only its publicly available API (i. e., respecting the "encapsulation principle"). This approach is applicable in automatic test generation tools to create correct and easy to maintain test cases. It is based on an original search algorithm, augmented with symbolic execution, which attempts to find an action sequence that creates the objects needed to exercise interesting executions found by the testing tool. The approach targets the JVM platform and considers several JVM-specific features but is general enough and can be extended to other programming languages.

The proposed approach was implemented as a module in an automatic testing tool provided in Kex platform and evaluated on a set of open-source projects. The results show it can successfully and efficiently generate 64.4 % of target objects on average, with the best result being 99.9 %, in a reasonable time. Based on the results, we can say Reanimator is applicable for automatic test case snippet generation.

In the future, we plan to explore the following directions:

- perform a more thorough investigation of Reanimator failures to understand how we can increase its success rate;
- improve support of complex types such as (unordered) collections and/or try using other symbolic execution engines;
- support generation of higher order functions;
- integrate our approach with unit-testing engines to generate additional assertions.

References

1. **Ayewah N., Pugh W., Hovemeyer D.** et al. Using static analysis to find bugs, *IEEE software,* 2008, vol. 25, no. 5, pp. 22—29. DOI: 10.1109/MS.2008.130.
2. **Calcagno C., Distefano D., Dubreil J.** et al. Moving fast with software verification, *NASA Formal Methods Symposium,* Springer, 2015, LNPSE, vol. 9058, pp. 3—11. DOI: 10.1007/978-3-319-17524-9_1.
3. **Sharma R. M.** Quantitative analysis of automation and manual testing, *International Journal of Engineering and Innovative Technology*, 2014, vol. 4, no. 1, pp. 252—257.
4. **Klees G., Ruef A., Cooper B.** et al. Evaluating fuzz testing, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 2123—2138. DOI: 10.1145/3243734.3243804.
5. **Braione P., Denaro G., Pezze M.** JBSE: A symbolic executor for Java programs with complex heap inputs, *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016, pp. 1018—1022. DOI: 10.1145/2950290.2983940.
6. **Fraser G., Arcuri A.** EvoSuite: automatic test suite generation for object-oriented software, *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, 2011, pp. 416—419. DOI: 10.1145/2025113.2025179.
7. **Csallner C., Smaragdakis Y.** JCrasher: an automatic robustness tester for Java, *Software: Practice and Experience*, 2004, vol. 34, no. 11, pp. 1025—1050. DOI: 10.1002/spe.602.
8. **Fraser G., Arcuri A.** Evolutionary generation of whole test suites, *2011 11th International Conference on Quality Software*, IEEE, 2011, pp. 31—40. DOI: 10.1109/QSIC.2011.19.
9. **Xie T., Marinov D., Schulte W., Notkin D.** Symstra: A framework for generating object-oriented unit tests using symbolic execution, *International Conference on Tools and Algorithms for the Construction and Analysis of Systems,* Springer, 2005, pp. 365—381. DOI: 10.1007/978-3-540-31980-1_24.
10. **Braione P., Denaro G., Mattavelli A., Pezze M.** SUSHI: A test generator for programs with complex structured inputs, *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*, IEEE, 2018, pp. 21—24. DOI: 10.1145/3183440.3183472.
11. **Demers F.-N., Malenfant J.** Reflection in logic, functional and object oriented programming: a short comparative study, *Proceedings of the IJCAI,* 1995, vol. 95, pp. 29—38.

12. **Orso A., Kennedy B.** Selective capture and replay of program executions, *ACM SIGSOFT Software Engineering Notes*, 2005, vol. 30, no. 4, pp. 1—7. DOI: 10.1145/1082983.1083251.

13. **Krikava F., Vitek J.** Tests from traces: automated unit test extraction for R, *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2018, pp. 232—241. DOI: 10.1145/3213846.3213863.

14. **Abdullin A. M., Itsykson V. M.** Kex: A platform for analysis of JVM programs, *Information and control systems,* 2022, no. 1 (116), pp. 30—43. DOI: 10.31799/1684-8853-2022-1-30-43.

15. **Braione P., Denaro G.** SUSHI and TARDIS at the SBST2019 Tool Competition, *IEEE/ACM 12th International Workshop on Search-Based Software Testing (SBST),* IEEE, 2019, pp. 25—28. DOI: 10.1109/SBST.2019.00016.

16. **Pouria D.** Well-informed Test Case Generation and Crash Reproduction, *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST),* IEEE, 2020, pp. 424—426. DOI: 10.1109/ICST46399.2020.00054.

17. **Gulwani S.** Dimensions in program synthesis, *Proceedings of the 12th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming*, 2010, pp. 13—24. DOI: 0.1145/1836089.1836091.

18. **Koushik S.** Concolic testing, *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering*, 2007, pp. 571—572. DOI: 10.1145/1321631.1321746.

19. **Elbaum S., Chin H. N., Dwyer M. B., Dokulil J.** Carving differential unit test cases from system test cases, *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering,* 2006, pp. 253—264. DOI: 10.1145/1181775.1181806.

20. **Burger M., Zeller A.** Minimizing reproduction of software failures, *Proceedings of the 2011 International Symposium on Software Testing and Analysis,* 2011, pp. 221—231. DOI: 10.1145/2001420.2001447.

21. **Jin W., Orso A.** Automated support for reproducing and debugging field failures, *ACM Transactions on Software Engineering and Methodology (TOSEM),* 2015, vol. 24, no. 4, pp. 26:1—26:35. DOI: 10.1145/2774218.

22. **Thanhofer-Pilisch J., Rabiser R., Krismayer T.** et al. An event-based capture-and-compare approach to support the evolution of systems of systems, *Proceedings of the 11th ACM International Conference on Distributed and Event-Based Systems*, 2017, pp. 261—270. DOI: 10.1145/3093742.3093909.

23. **Zamfir C., Candea G.** Execution synthesis: a technique for automated software debugging, *Proceedings of the 5th European Conference on Computer Systems,* 2010, pp. 321—334. DOI: 10.1145/1755913.1755946.

24. **Rößler J., Zeller A., Fraser G.** et al. Reconstructing core dumps, *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, IEEE, 2013, pp. 114—123. DOI: 10.1109/ICST.2013.18.

25. **Yu Tingting, Zaman Tarannum S., Wang Chao.** DESCRY: reproducing system-level concurrency failures, *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 694—704. DOI: DOI:10.1145/3106237.3106266.

26. **Leitner A., Pretschner A., Mori S.** et al. On the effectiveness of test extraction without overhead, *2009 International Conference on Software Testing Verification and Validation,* IEEE, 2009, pp. 416—425. DOI: DOI:10.1109/ICST.2009.30.

27. **Kifetew F. M., Jin W., Tiella R.** et al. Reproducing field failures for programs with complex grammar-based input, *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*, IEEE, 2014, pp. 163—172. DOI: 10.1109/ICST.2014.29.

28. **Barrett C., Tinelli C.** Satisfiability modulo theories, *Handbook of Model Checking,* Springer, 2018, pp. 305—343. DOI: 10.1007/978-3-319-10575-8_11.

29. **Jha Susmit, Limaye Rhishikesh, Seshia Sanjit A.** Beaver: Engineering an efficient SMT solver for bit-vector arithmetic, *International Conference on Computer Aided Verification,* Springer, 2009, pp. 668—674.

30. **Rummer P., Wahl T.** An SMT-LIB theory of binary floating-point arithmetic, *International Workshop on Satisfiability Modulo Theories (SMT),* 2010, pp. 151.

31. **Kapus T., Cadar C.** A segmented memory model for symbolic execution, *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 774—784. DOI: 10.1145/3338906.3338936.

32. **Stump A., Barrett C. W., Dill D. L., Levitt J.** A decision procedure for an extensional theory of arrays, *Proceedings 16th Annual IEEE Symposium on Logic in Computer Science,* IEEE, 2001, pp. 29—37.

33. **Clarke E., Biere A., Raimi R., Zhu Y.** Bounded model checking using satisfiability solving, *Formal Methods in System Design*, 2001, vol. 19, no. 1, pp. 7—34. DOI: 10.1023/A:1011276507260.

34. **Godefroid P., Levin M. Y., Molnar D. A.** Automated Whitebox Fuzz Testing, *NDSS,* 2008, vol. 8, pp. 151—166.

35. **Brummayer R., Biere A.** Boolector: An efficient SMT solver for bit-vectors and arrays, *International Conference on Tools and Algorithms for the Construction and Analysis of Systems* 2009., Springer, 2009, LNTCS, vol. 5505, pp. 174—177. DOI: 10.1007/978-3-642-00768-2_16.

36. **De Moura L., Bjørner N.** Z3: An efficient SMT solver, *International Conference on Tools and Algorithms for the Construction and Analysis of Systems 2008,* Springer-Verlag, 2008, LNTCS, vol. 4963, pp. 337—340. DOI: 10.1007/978-3-540-78800-3_24.

37. **Vijay G., Trevor H.** STP constraint solver: Simple theorem prover SMT solver, available at: https://stp.github.io/

38. **Devroey X., Panichella S., Gambi A.** Java Unit Testing Tool Competition: Eighth Round, *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, 2020, pp. 545—548. DOI: 10.1145/3387940.3392265.

УДК 519.684.6
DOI: 10.17587/prin.13.497-507

А. Ю. Морозов, канд. физ.-мат. наук, науч. сотр., morozov@infway.ru,
Федеральный исследовательский центр "Информатика и управление" Российской академии наук (ФИЦ ИУ РАН), Москва

Параллельный алгоритм параметрической идентификации динамических систем с интервальными параметрами

Представлен параллельный алгоритм параметрической идентификации динамических систем с интервальными параметрами. В основе этого алгоритма лежит ранее разработанный, обоснованный и апробированный алгоритм адаптивной интерполяции, который позволяет в явном виде получать зависимость состояний динамической системы от интервальных параметров. Решение задачи параметрической идентификации сводится к задаче минимизации определенной целевой функции в пространстве границ интервальных оценок параметров. За счет применения алгоритма адаптивной интерполяции при вычислении градиента целевой функции нет необходимости в дополнительном анализе и моделировании исходной динамической системы, поэтому для оптимизации удобно использовать методы первого порядка. Однако задача вычисления целевой функции и градиента включает в себя набор задач условной минимизации для явных функций, решение которых можно выполнить независимо друг от друга. Рассмотрены основные аспекты и особенности распараллеливания и реализации алгоритма параметрической идентификации и выполнена его апробация на нескольких представительных примерах. Проанализированы ускорение и эффективность распараллеливания.

Ключевые слова: *распараллеливание, OpenMP, алгоритм параметрической идентификации, алгоритм адаптивной интерполяции, многомерная интерполяция, интервальные системы обыкновенных дифференциальных уравнений, градиентные методы, оптимизация*

Введение

Обратные задачи играют важную роль в разных современных прикладных областях [1—3]. Задача параметрической идентификации возникает на этапе, когда математическая модель процесса уже определена, но неизвестными остаются ее параметры, которые необходимо подобрать так, чтобы модель наилучшим образом воспроизводила имеющиеся экспериментальные данные.

Данные, полученные во время эксперимента, как правило имеют некоторый разброс, поэтому при построении математической модели обычно преследуется цель минимизации отклонения модельных значений от экспериментальных.

Выделим работы [4—6], посвященные интервальному аппарату. Ключевое отличие интервальных моделей от обычных заключается в том, что они дают интервальные оценки на интересующие величины ("коридоры" возможных значений). Поэтому в случае с параметрической идентификацией основная цель — нахождение таких интервальных параметров, при которых модельные интервальные оценки будут полностью покрывать экспериментальные данные (или минимизировать отклонение от них).

Существует ряд работ, посвященных интервальным обратным задачам [7—9]. В основном в них рассмотрены математические модели определенного класса, например линейные. В настоящей же работе описан подход [10], позволяющий работать с нелинейными динамическими системами, заданными в общем виде.

В основе подхода лежит ранее разработанный алгоритм адаптивной интерполяции для моделирования динамических систем с интервальными параметрами [11, 12]. Цель алгоритма заключается в построении для каждого момента времени кусочно-полиномиальной функции, интерполирующей зависимость состояния динамической системы от неизвестных параметров, или, другими словами, параметрического множества состояний.

Данный алгоритм относится к методам, восстанавливающим явную зависимость интересующих величин от интервальных параметров (в этой группе выделяются символьные методы [13—15] и полиномиальные методы [16, 17]). Для задач с большим числом неопределенностей были разработаны две модификации алгоритма [12, 18] на основе разреженных сеток [19—21] и тензорных поездов [22, 23]. В работах [24—26] рассмотрены различные аспекты и особенности распараллеливания и реализации алгоритма с использованием технологий CUDA [27] и OpenMP [28].

Идея описываемого подхода [10] к решению задач параметрической идентификации заключается в построении и минимизации целевой функции, зависящей от границ интервальных оценок параметров системы и характеризующей расстояние между параметрическими множествами состояний и соответствующими экспериментальными данными.

Для вычисления значения целевой функции необходимо:

1) выполнить моделирование динамической системы с интервальными параметрами;

2) вычислить расстояние между полученными параметрическими множествами и экспериментальными данными.

Первый пункт подразумевает применение алгоритма адаптивной интерполяции. В процессе его работы требуется многократно выполнять моделирование исходной системы с определенными значениями неизвестных параметров, что является независимыми подзадачами, которые можно решать одновременно друг с другом. Поэтому данная задача имеет высокий потенциал распараллеливания.

Второй пункт заключается в решении для каждого момента времени, в котором известна экспериментальная информация о фазовых переменных, задачи поиска расстояния между параметрическим множеством и исходными данными. В этом случае тоже речь идет о независимых подзадачах, которые можно решать параллельно.

В качестве технологии для распараллеливания вычислений используется OpenMP (*Open Multi-Processing*), так как она позволяет достаточно легко создавать многопоточные приложения на различных языках программирования. Программный комплекс OpenMP — это набор директив компилятора, библиотечных процедур и переменных окружения, которые предназначены для программирования многопоточных приложений на многопроцессорных системах с общей памятью [29]. Выбор технологии OpenMP обусловлен неоднородностью независимых подзадач, а также простотой и эффективностью данной технологии.

Постановка задачи

Будем рассматривать автономные дискретные динамические системы с интервальным начальным состоянием и интервальными параметрами:

$$\begin{cases} y_i^{k+1} = F_i\left(y_1^k,\ y_2^k,\ ...,\ y_n^k,\ \theta_1,\ \theta_2,\ ...,\ \theta_m\right), \\ y_i^0 \in \left[\underline{y_i^0},\ \overline{y_i^0}\right],\ i = 1,...,n, \\ \theta_j \in \left[\underline{\theta_j},\ \overline{\theta_j}\right],\ j = 1,...,m, \\ k = 0,\ ...,\ N-1, \end{cases} \tag{1}$$

где n — размерность фазового пространства; m — число параметров; k — дискретное время; N — конечный момент времени; $\mathbf{y}^k = \left(y_1^k,\ y_2^k,\ ...,\ y_n^k\right)^{\mathrm{T}}$ — вектор состояния системы в момент времени k; $\boldsymbol{\theta} = \left(\theta_1, \theta_2,\ ...,\ \theta_m\right)^{\mathrm{T}}$ — вектор параметров системы; $\underline{y_i^0} \leqslant \overline{y_i^0},\ i = 1,\ ...,\ n,\ \underline{\theta_j} \leqslant \overline{\theta_j},\ j = 1,\ ...,\ m$ — нижние и верхние границы интервальных неопределенностей; $\mathbf{F} = \left(F_1,\ F_2,\ ...,\ F_n\right)^{\mathrm{T}}$ — вектор-функция, которая определяет переход системы из одного состояния в другое.

Динамическая система может быть задана в виде системы обыкновенных дифференциальных уравнений (ОДУ). В этом случае $\mathbf{F}\left(\mathbf{y}^k,\ \boldsymbol{\theta}\right) = \mathbf{u}\left(t_{k+1}\right)$, где $\mathbf{u}\left(t_{k+1}\right)$ — решение соответствующей системы ОДУ в конечный момент времени:

$$\begin{cases} \dfrac{du_i(t)}{dt} = f_i\left(u_1(t),\ u_2(t),\ ...,\ u_n(t),\ \theta_1,\ \theta_2,\ ...,\ \theta_m\right), \\ u_i\left(t_k\right) = y_i^k, \\ i = 1,\ ...,\ n, \\ t \in \left[t_k,\ t_{k+1}\right], \end{cases}$$

у которой правая часть $\mathbf{f} = \left(f_1,\ f_2,\ ...,\ f_n\right)^{\mathrm{T}}$ удовлетворяет всем условиям, обеспечивающим единственность и существование решения при всех $u\left(t_0\right) \in \left[\underline{y_i^0},\ \overline{y_i^0}\right]$, $i = 1,\ ...,\ n$ и $\theta_j \in \left[\underline{\theta_j},\ \overline{\theta_j}\right]$, $j = 1,\ ...,\ m$.

В частности, $\mathbf{F}\left(\mathbf{y}^k,\ \boldsymbol{\theta}\right)$ может реализовывать M шагов правила трех восьмых (четвертый порядок аппроксимации) [30]:

$$\begin{aligned} &\mathbf{F}\left(\mathbf{y}^k,\ \boldsymbol{\theta}\right) = \mathbf{u}^M, \\ &\mathbf{u}^M = \mathbf{u}^{M-1} + \frac{h}{8}\left(\mathbf{K}_1^{M-1} + 3\mathbf{K}_2^{M-1} + 3\mathbf{K}_3^{M-1} + \mathbf{K}_4^{M-1}\right), \\ &\mathbf{u}^0 = \mathbf{y}^k, \\ &\mathbf{K}_1^{M-1} = \mathbf{f}\left(\mathbf{u}^{M-1},\ \boldsymbol{\theta}\right), \\ &\mathbf{K}_2^{M-1} = \mathbf{f}\left(\mathbf{u}^{M-1} + \frac{h}{3}\mathbf{K}_1^{M-1},\ \boldsymbol{\theta}\right), \\ &\mathbf{K}_3^{M-1} = \mathbf{f}\left(\mathbf{u}^{M-1} + h\left[\mathbf{K}_2^{M-1} - \frac{1}{3}\mathbf{K}_1^{M-1}\right],\ \boldsymbol{\theta}\right), \\ &\mathbf{K}_4^{M-1} = \mathbf{f}\left(\mathbf{u}^{M-1} + h\left[\mathbf{K}_1^{M-1} - \mathbf{K}_2^{M-1} + \mathbf{K}_3^{M-1}\right],\ \boldsymbol{\theta}\right), \end{aligned}$$

где $h = \left(t_{k+1} - t_k\right)/M$.

Состояние системы (1) в каждый момент времени является параметрическим множеством:

$$Y^k = \left\{ \mathbf{y}^k\left(x_1,\, x_2,\, ...,\, x_n,\, z_1,\, z_2,\, ...,\, z_m\right) \middle| \begin{matrix} x_i \in \left[\underline{y_i^0},\, \overline{y_i^0}\right],\ i = 1,\, ...,\, n, \\ z_j \in \left[\underline{\theta_j},\, \overline{\theta_j}\right],\ j = 1,\, ...,\, m \end{matrix} \right\}, \tag{2}$$

где $\mathbf{y}^k\left(x_1,\, x_2,\, ...,\, x_n,\, z_1,\, z_2,\, ...,\, z_m\right)$ задается следующей рекуррентной формулой:

$$\begin{aligned} &\mathbf{y}^k\left(\mathbf{x},\, \mathbf{z}\right) = \mathbf{F}\left(\mathbf{y}^{k-1}\left(\mathbf{x},\, \mathbf{z}\right),\, \mathbf{z}\right), \\ &\mathbf{y}^0\left(\mathbf{x},\, \mathbf{z}\right) = \mathbf{x}, \end{aligned} \tag{3}$$

где $\mathbf{x} = \left(x_1,\, x_2,\, ...,\, x_n\right)^{\mathrm{T}}$ и $\mathbf{z} = \left(z_1,\, z_2,\, ...,\, z_m\right)^{\mathrm{T}}$.

Пусть известна $(N+1)$ экспериментальная точка в фазовом пространстве в различные моменты времени:

$$\hat{\mathbf{y}}^k = \left(\hat{y}_1^k,\, \hat{y}_2^k,\, ...,\, \hat{y}_n^k\right)^{\mathrm{T}},\ k = 0,\, ...,\, N. \tag{4}$$

Задача параметрической идентификации заключается в нахождении таких границ интервальных оценок $\underline{y_i^0} \leqslant y_i^0$, $i = 1,\, ...,\, n$, $\theta_j \leqslant \overline{\theta_j}$, $j = 1,\, ...,\, m$, при которых множество (2) будет содержать в себе все экспериментальные точки (4) или минимизировать отклонение от них. Отметим, что часть начальных условий и параметров может быть априори известна, в этом случае они не нуждаются в идентификации. Однако для удобства изложения будем предполагать, что все границы интервальных оценок являются неизвестными.

Алгоритм параметрической идентификации

Для нахождения интервальных оценок выполняется переход к задаче минимизации расстояния между модельными множествами состояний (2) системы (1) и соответствующими экспериментальными точками (4). Минимизируется целевая функция

$$\begin{aligned} J\left(\underline{y_1^0},\, \overline{y_1^0},\, ...,\, \underline{y_n^0},\, \overline{y_n^0},\, \underline{\theta_1},\, \overline{\theta_1},\, ...,\, \underline{\theta_m},\, \overline{\theta_m}\right) = \\ = \sum_{k=0}^{N} \rho\left(Y^k,\, \hat{\mathbf{y}}^k\right), \end{aligned} \tag{5}$$

где

$$\rho\left(Y^k,\, \hat{\mathbf{y}}^k\right) = \min_{\mathbf{y}^k \in Y^k} \left\| \mathbf{y}^k - \hat{\mathbf{y}}^k \right\|^2 \tag{6}$$

— квадрат расстояния между множеством Y^k и точкой $\hat{\mathbf{y}}^k$.

В качестве нормы удобно использовать евклидову норму. Важную роль при построении градиента целевой функции (5) играют прообразы соответствующих точек минимумов (6) в пространстве начальных условий и параметров.

Запишем (6) следующим образом:

$$\begin{aligned} \rho\left(Y^k,\, \hat{\mathbf{y}}^k\right) = \min_{\substack{x_i \in \left[\underline{y_i^0},\, \overline{y_i^0}\right],\ i=1,\,...,\,n, \\ z_i \in \left[\underline{\theta_j},\, \overline{\theta_j}\right],\ j=1,\,...,\,m.}} J^k\left(x_1,\, x_2,\, ...,\, x_n,\right. \\ \left. z_1,\, z_2,\, ...,\, z_m\right), \end{aligned} \tag{7}$$

где

$$\begin{aligned} J^k\left(x_1,\, x_2,\, ...,\, x_n,\, z_1,\, z_2,\, ...,\, z_m\right) = \\ = \sum_{i=1}^{n} \left[y_i^k\left(x_1,\, x_2,\, ...,\, x_n,\, z_1,\, z_2,\, ...,\, z_m\right) - \hat{y}_i^k \right]^2. \end{aligned} \tag{8}$$

Отметим, что ключевым моментом является то, что $\mathbf{y}^k\left(x_1,\, x_2,\, ...,\, x_n,\, z_1,\, z_2,\, ...,\, z_m\right)$ можно получить в явном виде с помощью алгоритма адаптивной интерполяции, и при этом нет необходимости в применении рекуррентной формулы (3), которая эквивалентна моделированию исходной системы (1).

Обозначим через $\left(\tilde{\mathbf{x}}^k,\, \tilde{\mathbf{z}}^k\right) = \left(\tilde{x}_1^k,\, \tilde{x}_2^k,\, ...,\, \tilde{x}_n^k,\, \tilde{z}_1^k,\, \tilde{z}_2^k,\, ...,\, \tilde{z}_m^k\right)^{\mathrm{T}}$ точки минимума (7):

$$\begin{aligned} \left(\tilde{x}_1^k,\, \tilde{x}_2^k,\, ...,\, \tilde{x}_n^k,\, \tilde{z}_1^k,\, \tilde{z}_2^k,\, ...,\, \tilde{z}_m^k\right)^{\mathrm{T}} = \\ = \arg\min_{\substack{x_i \in \left[\underline{y_i^0},\, \overline{y_i^0}\right],\, i=1,\,...,\,n, \\ z_i \in \left[\underline{\theta_j},\, \overline{\theta_j}\right],\ j=1,\,...,\,m.}} J^k\left(x_1,\, x_2,\, ...,\, x_n,\, z_1,\, z_2,\, ...,\, z_m\right). \end{aligned} \tag{9}$$

Запишем производные для (8):

$$\begin{aligned} &\frac{dJ^k\left(\mathbf{x},\, \mathbf{z}\right)}{dx_i} = 2\sum_{l=1}^{n} \left[y_l^k\left(\mathbf{x},\, \mathbf{z}\right) - y_l^k \right] \frac{dy_l^k\left(\mathbf{x},\, \mathbf{z}\right)}{dx_i}, \\ &i = 1,\, ...,\, n, \\ &\frac{dJ^k\left(\mathbf{x},\, \mathbf{z}\right)}{dz_j} = 2\sum_{l=1}^{n} \left[y_l^k\left(\mathbf{x},\, \mathbf{z}\right) - y_l^k \right] \frac{dy_l^k\left(\mathbf{x},\, \mathbf{z}\right)}{dz_j}, \\ &j = 1,\, ...,\, m. \end{aligned} \tag{10}$$

Так как $\left(\tilde{x}_1^k,\, \tilde{x}_2^k,\, ...,\, \tilde{x}_n^k,\, \tilde{z}_1^k,\, \tilde{z}_2^k,\, ...,\, \tilde{z}_m^k\right)^{\mathrm{T}}$ являются точками минимума, то

$$\frac{dJ^k\left(\tilde{x}_1^k,\ \tilde{x}_2^k,\ ...,\ \tilde{x}_n^k,\ \tilde{z}_1^k,\ \tilde{z}_2^k,\ ...,\ \tilde{z}_m^k\right)}{dx_i}\begin{cases}\leqslant 0,\ \tilde{x}_i^k = \overline{y_i^0},\\ = 0,\ \tilde{x}_i^k \in \left(\underline{y_i^0}, \overline{y_i^0}\right),\ i = 1,\ ...,\ n,\\ \geqslant 0,\ \tilde{x}_i^k = \underline{y_i^0},\end{cases}$$

$$\frac{dJ^k\left(\tilde{x}_1^k,\ \tilde{x}_2^k,\ ...,\ \tilde{x}_n^k,\ \tilde{z}_1^k,\ \tilde{z}_2^k,\ ...,\ \tilde{z}_m^k\right)}{dz_j}\begin{cases}\leqslant 0,\tilde{z}_j^k = \overline{\theta_j},\\ = 0,\tilde{z}_j^k \in \left(\underline{\theta_j},\ \overline{\theta_j}\right),\ j = 1,\ ...,\ m.\\ \geqslant 0,\tilde{z}_j^k = \underline{\theta_j},\end{cases} \tag{11}$$

Если точки $\left(\tilde{x}_1^k,\ \tilde{x}_2^k,\ ...,\ \tilde{x}_n^k,\ \tilde{z}_1^k,\ \tilde{z}_2^k,\ ...,\ \tilde{z}_m^k\right)^{\mathrm{T}}$ лежат внутри области неопределенности

$$\chi = \left[\underline{y_1^0},\ \overline{y_1^0}\right]\times\left[\underline{y_2^0},\ \overline{y_2^0}\right]\times ...\times\left[\underline{y_n^0},\ \overline{y_n^0}\right]\times\left[\underline{\theta_1},\ \overline{\theta_1}\right]\times\left[\underline{\theta_2},\ \overline{\theta_2}\right]\times ...\times\left[\underline{\theta_m},\ \overline{\theta_m}\right],$$

то производные целевой функции по $\underline{y_1^0},\ \overline{y_1^0},\ ...,\ \underline{y_n^0},\ \overline{y_n^0},\ \underline{\theta_1},\ \overline{\theta_1},\ ...,\ \underline{\theta_m},\ \overline{\theta_m}$ будут равны нулю, а если на границе $\partial\chi$, то — соответствующим производным (11). С учетом (10) и (11) в результате получаем следующие компоненты градиента для целевой функции (5):

$$\begin{gathered}\frac{dJ}{d\underline{y_i^0}} = 2\sum_{k=0}^{N}\max\left(0,\ \sum_{l=1}^{n}\left[y_l^k\left(\tilde{\mathbf{x}}^k,\ \tilde{\mathbf{z}}^k\right) - y_l^k\right]\frac{dy_l^k\left(\tilde{\mathbf{x}}^k,\ \tilde{\mathbf{z}}^k\right)}{dx_i}\right),\\ \frac{dJ}{d\overline{y_i^0}} = 2\sum_{k=0}^{N}\min\left(0,\ \sum_{l=1}^{n}\left[y_l^k\left(\tilde{\mathbf{x}}^k,\ \tilde{\mathbf{z}}^k\right) - y_l^k\right]\frac{dy_l^k\left(\tilde{\mathbf{x}}^k,\ \tilde{\mathbf{z}}^k\right)}{dx_i}\right),\ i = 1,\ ...,\ n,\\ \frac{dJ}{d\underline{\theta_j}} = 2\sum_{k=0}^{N}\max\left(0,\ \sum_{l=1}^{n}\left[y_l^k\left(\tilde{\mathbf{x}}^k,\ \tilde{\mathbf{z}}^k\right) - y_l^k\right]\frac{dy_l^k\left(\tilde{\mathbf{x}}^k,\ \tilde{\mathbf{z}}^k\right)}{dz_j}\right),\\ \frac{dJ}{d\overline{\theta_j}} = 2\sum_{k=0}^{N}\min\left(0,\ \sum_{l=1}^{n}\left[y_l^k\left(\tilde{\mathbf{x}}^k,\ \tilde{\mathbf{z}}^k\right) - y_l^k\right]\frac{dy_l^k\left(\tilde{\mathbf{x}}^k,\ \tilde{\mathbf{z}}^k\right)}{dz_j}\right),\ j = 1,\ ...,\ m.\end{gathered} \tag{12}$$

Так как согласно (12) ширина получающихся интервальных неопределенностей не будет уменьшаться, то дополнительно определим правила сужения интервалов:

$$\begin{gathered}\underline{y_i^0} = \min_{k=0,...,N}\left(\tilde{x}_i^k\right),\ \text{если}\ \frac{dJ}{d\underline{y_i^0}} = 0,\\ \overline{y_i^0} = \max_{k=0,...,N}\left(\tilde{x}_i^k\right),\ \text{если}\ \frac{dJ}{d\overline{y_i^0}} = 0,\ i = 1,\ ...,\ n,\\ \underline{\theta_j} = \min_{k=0,...,N}\left(\tilde{z}_j^k\right),\ \text{если}\ \frac{dJ}{d\underline{\theta_j}} = 0,\\ \overline{\theta_j} = \max_{k=0,...,N}\left(\tilde{z}_j^k\right),\ \text{если}\ \frac{dJ}{d\overline{\theta_j}} = 0,\ j = 1,\ ...,\ m.\end{gathered} \tag{13}$$

Далее для поиска неизвестных границ интервальных оценок можно использовать методы оптимизации первого порядка [31]. Одна итерация метода градиентного спуска записывается следующим образом:

$$\underline{y_i^0}^{(it+1)} = \underline{y_i^0}^{(it)} - \lambda^{(it)}\frac{dJ\left(\underline{y_1^0}^{(it)}, \overline{y_1^0}^{(it)}, ...,\ \underline{\theta_1}^{(it)}, \overline{\theta_1}^{(it)}, ...\right)}{d\underline{y_i^0}},$$

$$\overline{y_i^0}^{(it+1)} = \overline{y_i^0}^{(it)} - \lambda^{(it)}\frac{dJ\left(\underline{y_1^0}^{(it)}, \overline{y_1^0}^{(it)}, ...,\ \underline{\theta_1}^{(it)}, \overline{\theta_1}^{(it)}, ...\right)}{d\overline{y_i^0}},$$

$$i = 1,\ ...,\ n,$$

$$\underline{\theta_j}^{(it+1)} = \underline{\theta_j}^{(it)} - \lambda^{(it)}\frac{dJ\left(\underline{y_1^0}^{(it)}, \overline{y_1^0}^{(it)}, ...,\ \underline{\theta_1}^{(it)}, \overline{\theta_1}^{(it)}, ...\right)}{d\underline{\theta_j}},$$

$$\overline{\theta_j}^{(it+1)} = \overline{\theta_j}^{(it)} - \lambda^{(it)}\frac{dJ\left(\underline{y_1^0}^{(it)}, \overline{y_1^0}^{(it)}, ...,\ \underline{\theta_1}^{(it)}, \overline{\theta_1}^{(it)}, ...\right)}{d\overline{\theta_j}},$$

$$j = 1,...,m,$$

где $\lambda^{(it)}$ — скорость градиентного спуска.

После каждой итерации применяются правила (13). Начальное приближение $\underline{y_i^0}^{(0)} < \overline{y_i^0}^{(0)}$, $\underline{\theta_j}^{(0)} < \overline{\theta_j}^{(0)}$ задается произвольным образом, однако оно может сразу удовлетворить условию $\hat{\mathbf{y}}^0 \in Y^0$. Процесс поиска останавливается, когда $J \leqslant \varepsilon$ или когда $\|\nabla J\| \leqslant \delta$, где ε и δ — наперед заданные положительные числа, характеризующие требуемую точность.

Распараллеливание и реализация

Для того чтобы распараллелить работу алгоритма, его достаточно представить в виде набора независимых подзадач.

В рамках одной итерации градиентного спуска необходимо решить перечисленные далее задачи.

1. Решить прямую задачу (1). Для этого применяется параллельный алгоритм адаптивной интерполяции на основе разреженных сеток с нелинейным базисом, описанный в работе [25]. В результате будет получена вектор-функция $\mathbf{P}^k(\mathbf{x}, \mathbf{z})$, интерполирующая $\mathbf{y}^k(\mathbf{x}, \mathbf{z})$, $k = 0, ..., N$. В выражение (12) вместо $\mathbf{y}^k(\mathbf{x}, \mathbf{z})$ подставляется $\mathbf{P}^k(\mathbf{x}, \mathbf{z})$, а вместо $\frac{dy_l^k(\mathbf{x}, \mathbf{z})}{dx_i}$ и $\frac{dy_l^k(\mathbf{x}, \mathbf{z})}{dz_j}$ подставляются $\frac{dP_l^k(\mathbf{x}, \mathbf{z})}{dx_i}$ и $\frac{dP_l^k(\mathbf{x}, \mathbf{z})}{dz_j}$, $l = 1, ..., n$, $i = 1, ..., n$, $j = 1, ..., m$, которые вычисляются аналитически.

2. Решить $(N+1)$ задач (9) поиска минимума для явной функции:

$$\left(\tilde{\mathbf{x}}^k, \tilde{\mathbf{z}}^k\right)^{\mathrm{T}} = \underset{\substack{x_i \in \left[\underline{y_i^0}, \overline{y_i^0}\right],\ i=1,\ldots,n,\\ z_i \in \left[\underline{\theta_j}, \overline{\theta_j}\right],\ j=1,\ldots,m.}}{\arg\min} \sum_{i=1}^{n} \left[P_i^k(\mathbf{x}, \mathbf{z}) - \hat{y}_i^k\right]^2. \quad (14)$$

Данные задачи являются независимыми друг от друга, поэтому их можно решать одновременно. Отметим, что в зависимости от исходной системы (1) для разных k может потребоваться разное вычислительное время. Поэтому при распараллеливании соответствующего цикла с помощью OpenMP необходимо

1. $\mathbf{P}^0 : \left[\underline{y_1^0}, \overline{y_1^0}\right] \times \ldots \times \left[\underline{y_n^0}, \overline{y_n^0}\right] \times \left[\underline{\theta_1}, \overline{\theta_1}\right] \times \ldots \times \left[\underline{\theta_m}, \overline{\theta_m}\right] \to \mathbb{R}^n$
2. $\mathbf{P}^0(\mathbf{x}, \mathbf{z}) = \mathbf{x}$
3. for $k = 1, ..., N$:
4. $\quad \mathbf{P}^k = \text{parallel } buildPolynomial\left(\mathbf{P}^{k-1}, \mathbf{F}\right)$
5. parallel for $k = 0, ..., N$:
6. $\quad \left(\tilde{\mathbf{x}}^k, \tilde{\mathbf{z}}^k\right) = \arg\min\left(\sum_{i=1}^{n}\left[P_i^k(\mathbf{x}, \mathbf{z}) - \hat{y}_i^k\right]^2\right)$
7. $\quad \frac{dJ^k}{dx_i} = 2\sum_{l=1}^{n}\left[P_l^k\left(\tilde{\mathbf{x}}^k, \tilde{\mathbf{z}}^k\right) - \hat{y}_l^k\right]\frac{dP_l^k\left(\tilde{\mathbf{x}}^k, \tilde{\mathbf{z}}^k\right)}{dx_i}, i = 1, ..., n,$
8. $\quad \frac{dJ^k}{dz_j} = 2\sum_{l=1}^{n}\left[P_l^k\left(\tilde{\mathbf{x}}^k, \tilde{\mathbf{z}}^k\right) - \hat{y}_l^k\right]\frac{dP_l^k\left(\tilde{\mathbf{x}}^k, \tilde{\mathbf{z}}^k\right)}{dz_j}, j = 1, ..., m.$
9. $J = \sum_{k=0}^{N}\sum_{i=1}^{n}\left[P_i^k\left(\tilde{\mathbf{x}}^k, \tilde{\mathbf{z}}^k\right) - \hat{y}_i^k\right]^2$
10. $\frac{dJ}{d\underline{y_i^0}} = 2\sum_{k=0}^{N}\max\left(0, \frac{dJ^k}{dx_i}\right), \frac{dJ}{d\overline{y_i^0}} = 2\sum_{k=0}^{N}\min\left(0, \frac{dJ^k}{dx_i}\right), i = 1, ..., n,$
11. $\frac{dJ}{d\underline{\theta_j}} = 2\sum_{k=0}^{N}\max\left(0, \frac{dJ^k}{dz_j}\right), \frac{dJ}{d\overline{\theta_j}} = 2\sum_{k=0}^{N}\min\left(0, \frac{dJ^k}{dz_j}\right), j = 1, ..., m.$

Рис. 1. Псевдокод вычисления значений целевой функции и ее градиента

применять директиву schedule(dynamic), чтобы потоки динамически распределяли задачи между собой.

На рис. 1 представлен псевдокод вычисления значения целевой функции и ее градиента. На входе: $\underline{y_i^0} \leqslant \overline{y_i^0}$, $\underline{\theta_j} \leqslant \overline{\theta_j}$, $i = 1, ..., n,$, $j = 1, ..., m$ — нижние и верхние границы интервальных неопределенностей. На выходе: J, $\frac{dJ}{d\underline{y_i^0}}$, $\frac{dJ}{d\overline{y_i^0}}$, $\frac{dJ}{d\underline{\theta_j}}$, $\frac{dJ}{d\overline{\theta_j}}$, $i = 1, ..., n$, $j = 1, ..., m$ — значение целевой функции и компонент градиента.

Функция *buildPolynomial* $\left(\mathbf{P}^{k-1}, \mathbf{F}\right)$ строит полином $\mathbf{P}^k(\mathbf{x}, \mathbf{z})$, который интерполирует $\mathbf{F}\left(\mathbf{P}^{k-1}(\mathbf{x}, \mathbf{z}), \mathbf{z}\right)$, и является основой алгоритма адаптивной интерполяции.

Начинать выполнять цикл papallel for $k = 0$, ..., N можно не ожидая завершения предыдущего цикла, необходимо только, чтобы по переменной k не было опережения.

Отметим, что задача (9) допускает множество решений, в этом случае с точки зрения ширины получающихся интервальных оценок целесообразно брать решение, находящееся ближе к центру текущей области неопределенности: $\left(\frac{\overline{y_1^0} + \underline{y_1^0}}{2}, ..., \frac{\overline{y_n^0} + \underline{y_n^0}}{2}, \frac{\overline{\theta_1} + \underline{\theta_1}}{2}, ..., \frac{\overline{\theta_m} + \underline{\theta_m}}{2}\right)^{\mathrm{T}}$.

На практике целесообразно найти несколько частных решений и взять ближайшее из них.

Результаты

Выполним решение задачи параметрической идентификации для двух систем ОДУ с интервальными неопределенностями. Параметры остановки метода градиентного спуска: $\varepsilon = \delta = 10^{-12}$. В качестве экспериментальных точек использовались квазиэкспериментальные точки (искусственно сгенерированные определенным образом точки):

$$\hat{\mathbf{y}}^k = \mathbf{P}^k\left(\hat{x}_1^k, \hat{x}_2^k, ..., \hat{x}_n^k, \hat{z}_1^k, \hat{z}_2^k, ..., \hat{z}_m^k\right),$$

где $\hat{x}_i^k = \mathrm{rand}\left[\underline{y_i^0}, \overline{y_i^0}\right]$, $i = 1, ..., n$; $\hat{z}_j^k = \mathrm{rand}\left[\underline{\theta_j}, \overline{\theta_j}\right]$, $j = 1, ..., m$ — равномерно распределенные на априори заданных интервалах случайные величины; $\mathbf{P}^k$ — решение прямой задачи, полученное с помощью алгоритма адаптивной интерполяции.

Для каждой задачи оцениваются ускорение $S = T_1/T_p$ и эффективность $E = S/P$, где T_1 — время работы последовательной версии алгоритма; T_p — время работы параллельной версии алгоритма с использованием P вычислительных потоков.

Характеристики используемой вычислительной машины: Intel(R) Xeon(R) CPU E5-2620 v4@2.10GHz, оперативная память 8x32 GiB DDR4 2666 MHz, режим работы процессора с памятью — двухканальный, число вычислительных ядер с гипертредингом — 8.

Вначале рассмотрим систему ОДУ с двумя интервальными параметрами:

$$\begin{cases} u' = -\dfrac{\alpha v}{1/2 + \sqrt{\sin^2(u) + v^2}}, \\ v' = \dfrac{\beta u}{1/2 + \sqrt{u^2 + \cos^2(v)}}, \\ u(0) = 5,\ v(0) = 0, \\ \alpha \in \left[\underline{\alpha}, \overline{\alpha}\right], \beta \in \left[\underline{\beta}, \overline{\beta}\right], \\ t \in [0, 60], \end{cases} \quad (15)$$

где $\underline{\alpha}$, $\overline{\alpha}$, $\underline{\beta}$, $\overline{\beta}$ — неизвестные границы интервальных параметров, которые подлежат определению.

Для генерации квазиэкспериментальных данных из интервала интегрирования были взяты 100 точек с постоянным шагом по времени и использовались значения параметров $\alpha \in [0,975, 1,025]$ и $\beta \in [0,900, 1,100]$. На рис. 2 показаны полученные точки на фазовой плоскости.

Начальное приближение в методе градиентного спуска: $\alpha^{(0)} \in [0,70, 0,75]$ и $\beta^{(0)} \in [0,70, 0,75]$. На рис. 3 продемонстрирован процесс решения задачи. Серым цветом показаны получающиеся модельные интервальные оценки фазовых переменных, а черным — экспериментальные данные. Найденные значения параметров $\alpha^{(29)} \in [0,975, 1,025]$ и $\beta^{(29)} \in [0,900, 1,100]$ совпадают с исходными.

На рис. 4 серым цветом показаны полученные параметрические множества Y^k, $k = 1, 3, 5, ..., 43$, на последней итерации градиентного спуска.

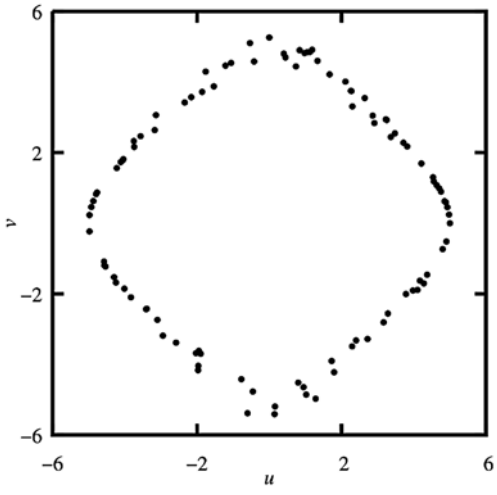


Рис. 2. Квазиэкспериментальные точки на фазовой плоскости

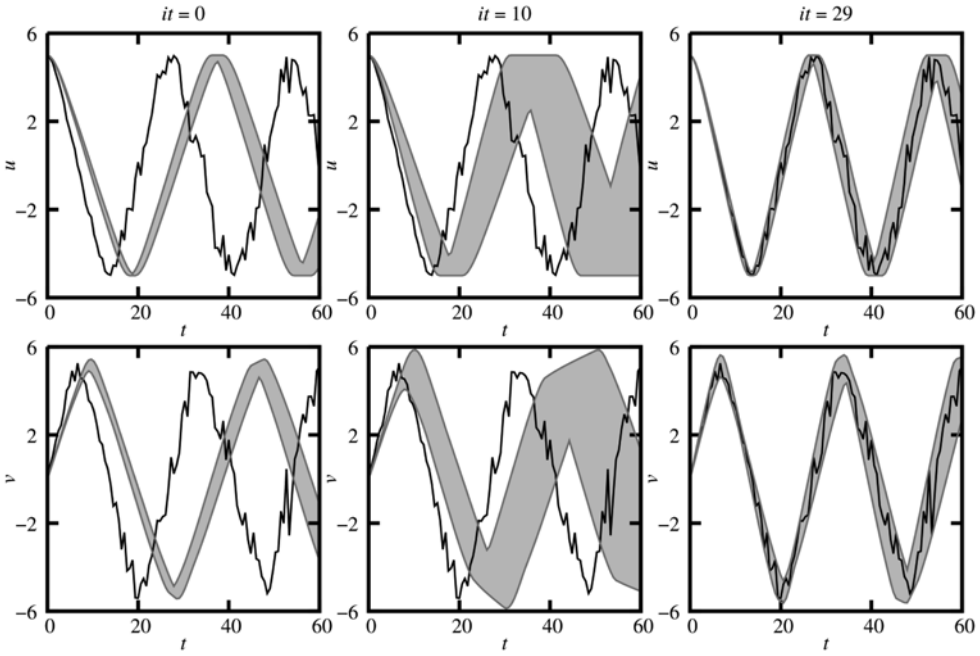


Рис. 3. Иллюстрация решения задачи параметрической идентификации системы (15)

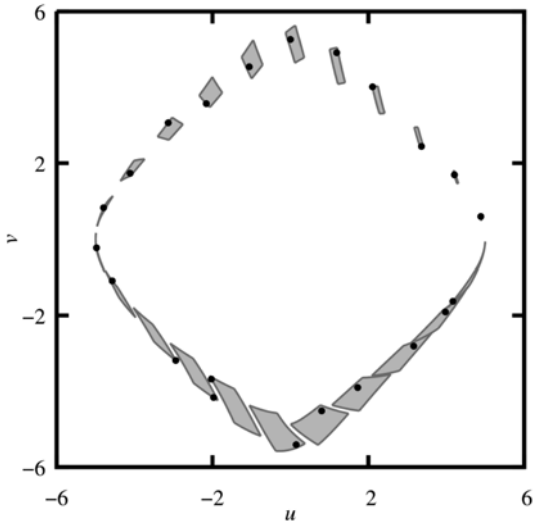


Рис. 4. Некоторые множества состояний системы (15) на 29-й итерации

На рис. 4 можно видеть, что все экспериментальные точки принадлежат соответствующим множествам.

На рис. 5 представлены графики зависимости ускорения S и эффективности E от числа вычислительных потоков P. За счет распараллеливания удалось сократить вычислительное время в 5 раз. Получение небольшого коэффициента распараллеливания связано, во-первых, с тем, что реальных ядер 8, во-вторых, с особенностями распараллеливания алгоритма адаптивной интерполяции, описанными в работе [25], а в-третьих, с неоднородностью задач (14) для разных k.

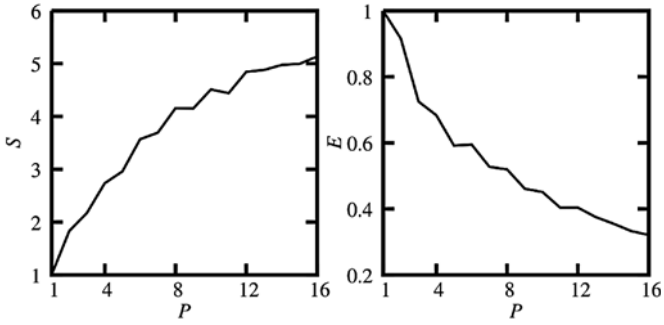


Рис. 5. Ускорение и эффективность распараллеливания при решении задачи параметрической идентификации системы (15)

Процесс вычисления вектор-функции $\mathbf{P}^k(\mathbf{x}, \mathbf{z})$ подразумевает частое обращение к памяти, и при числе потоков более 16 будет наблюдаться увеличение времени работы программы, связанное с особенностями взаимодействия с оперативной памятью.

Далее рассмотрим систему ОДУ, соответствующую модели осциллятора Дуффинга [32] с интервальными двумя начальными условиями и одним параметром:

$$\begin{cases} u' = v, \\ v' = u - \dfrac{1}{4}v - u^3 + \alpha\cos(\tau), \\ \tau' = 1, \\ \tau(0) = 0, \\ u(0) = u_0 \in \left[\underline{u_0}, \overline{u_0}\right], \\ v(0) = v_0 \in \left[\underline{v_0}, \overline{v_0}\right], \ \alpha \in \left[\underline{\alpha}, \overline{\alpha}\right], \\ t \in [0, 6]. \end{cases} \tag{16}$$

Как и в предыдущем примере, для генерации квазиэкспериментальных данных были взяты 100 точек с постоянным шагом по времени. Исходные значения начальных условий и параметра: $u_0 \in [1,90,\ 2,10]$, $v_0 \in [1,90,\ 2,10]$, $\alpha \in [0,27,\ 0,33]$. Начальное приближение: $u_0^{(0)} \in [1,50,\ 1,60]$, $v_0^{(0)} \in [1,60,\ 1,70]$, $\alpha^{(0)} \in [0,48,\ 0,78]$. На рис. 6 показаны зависимости интервальных оценок фазовых переменных от времени для различных итераций *it* градиентного спуска. На рис. 7 представлены полученные параметрические множества Y^k, $k = 1, 5, 9, ..., 97$, на фазовой плоскости и соответствующие экспериментальные точки.

На 33-й итерации полученные параметрические множества Y^k полностью содержат в себе экспериментальные точки: $J^{(33)} = 0$. Найденные интервальные оценки: $u_0^{(33)} \in [1,85,\ 2,11]$, $v_0^{(33)} \in [1,80,\ 2,37]$, $\alpha^{(33)} \in [0,46,\ 0,78]$. Их отличие от исходных оценок связано с многоэкстремальностью целевой функции.

На рис. 8 показаны графики зависимости ускорения S и эффективности E от числа вычислительных потоков P. Максимальное достигнутое ускорение — более 6 раз.

Алгоритм параметрической идентификации условно состоит из двух частей — решения прямой интервальной задачи и решения множества

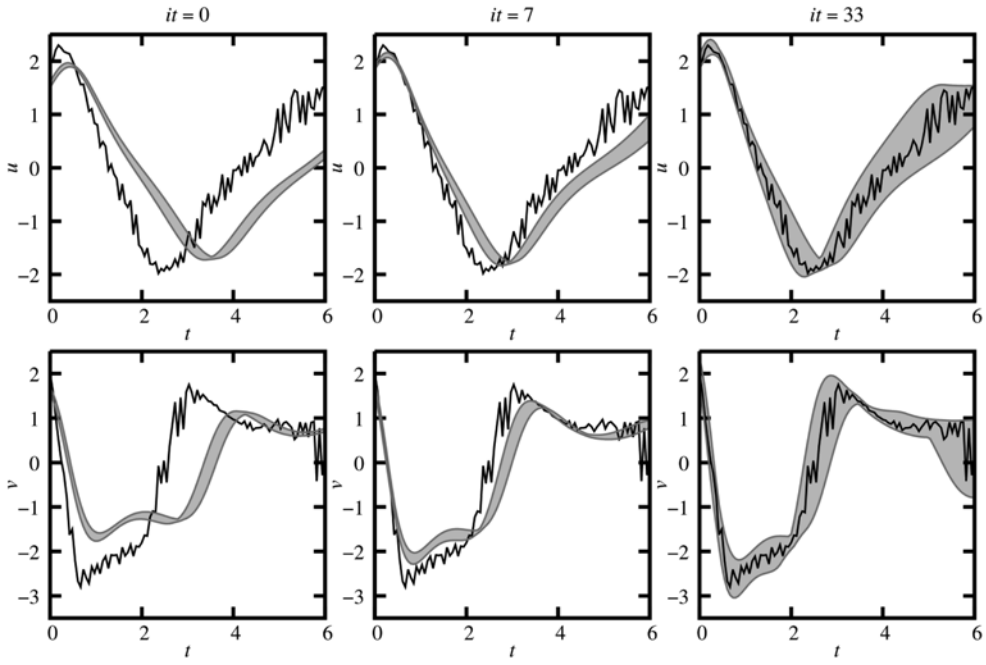


Рис. 6. Зависимости интервальных оценок фазовых переменных от времени в процессе параметрической идентификации системы (16)

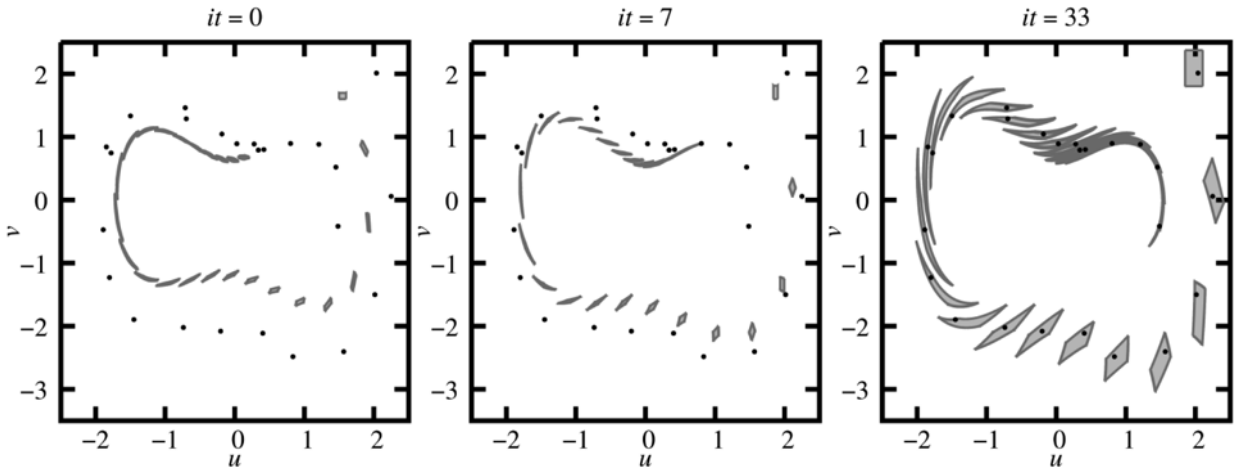


Рис. 7. Иллюстрация процесса решения задачи параметрической идентификации для системы (16) на фазовой плоскости

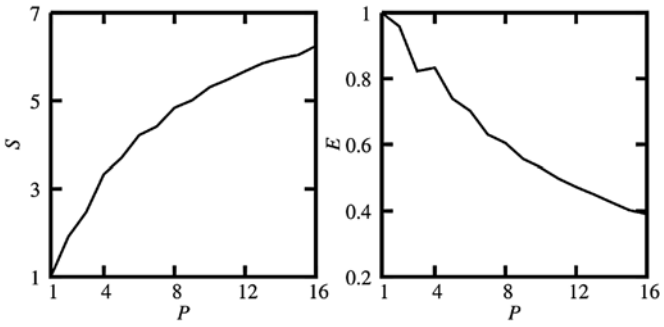


Рис. 8. Ускорение и эффективность распараллеливания при решении задачи параметрической идентификации системы (16)

задач минимизации для явных функций. Благодаря распараллеливанию данных операций можно существенно сократить время работы алгоритма.

Заключение

В работе рассматривается алгоритм параметрической идентификации интервальных динамических систем с позиции распараллеливания. Поиск неизвестных границ интервальных параметров сводится к решению задачи минимизации. При этом основные вычислительные затраты связаны с получением значения целевой функции и ее градиента. Вычисление целевой функции происходит в два этапа. Сначала решается прямая задача моделирования динамической системы с интервальными параметрами с помощью алгоритма адаптивной интерполяции. После этого решается множество независимых задач минимизации для явных функций. Оба этапа допускают распараллеливание. При апробации параллельной реализации алгоритма на двух системах ОДУ получено ускорение в 5—6 раз.

Список литературы

1. **Nenarokomov A. V., Alifanov O. M., Krainova I. V., Titov D. M., Morzhukhina A. V.** Estimation of environmental influence on spacecraft materials radiative properties by inverse problems technique // Acta Astronautica. 2019. Vol. 160. P. 323—330. DOI: 10.1016/j.actaastro.2019.04.014.

2. **Кабанихин С. И., Куликов И. М., Шишленин М. А.** Алгоритм восстановления характеристик начального состояния сверхновой звезды // Журнал вычислительной математики и математической физики. 2020. Том 60, № 6. С. 1035—1044. DOI: 10.31857/S0044466920060137.

3. **Абгарян К. К., Носков Р. Г., Ревизников Д. Л.** Обратная коэффициентная задача теплопереноса в слоистых наноструктурах // Известия высших учебных заведений. Материалы электронной техники. 2017. Том 20, № 3. С. 213—219. DOI: 10.17073/1609-3577-2017-3-213-219.

4. **Moore R. E., Kearfott R. B., Cloud M. J.** Introduction to Interval Analysis, SIAM, 2009. 223 p.

5. **Шарый С. П.** Конечномерный интервальный анализ. Новосибирск: Институт вычислительных технологий СО РАН. Изд. XYZ, 2022. 653 с.

6. **Добронец Б. С.** Интервальная математика. Красноярск: Краснояр. гос. ун-т, 2004. 216 с.

7. **Дилигенская А. Н., Самокиш А. В.** Параметрическая идентификация в обратных задачах теплопроводности в условиях интервальной неопределенности на основе нейронных сетей // Вестник Самарского государственного технического университета. 2020. Т. 28. № 4 (68). С. 6—18.

8. **Петрикевич Я. И.** Структурно-параметрическая идентификация динамических объектов по интервальным исходным данным: дис. ... канд. техн. наук: 05.13.18. Кемерово: Кемеровский государственный университет, 2006. 225 с.

9. **Xiao N., Fedele F., Muhanna R. L.** Inverse Problems Under Uncertainties-An Interval Solution for the Beam Finite Element // Conference Paper: 11th International Conference on Structural Safety & Reliability, New York, NY, USA, 2013. 8 p. URL: https://www.researchgate.net/publication/269518192. DOI: 10.1201/b16387-430.

10. **Морозов А. Ю., Ревизников Д. Л.** Интервальный подход к решению задач параметрической идентификации динамических систем // Дифференциальные уравнения. 2022. Том 58, № 7. С. 962—976. DOI: 10.31857/S0374064122070081.

11. **Морозов А. Ю., Ревизников Д. Л., Гидаспов В. Ю.** Алгоритм адаптивной интерполяции на основе kd-дерева для решения задач химической кинетики с интервальными параметрами // Математическое моделирование. 2018. Т. 30. № 12. С. 129—144. DOI: 10.31857/S023408790001940-8.

12. **Морозов А. Ю., Ревизников Д. Л.** Алгоритм адаптивной интерполяции на разреженных сетках для численного интегрирования систем обыкновенных дифференциальных уравнений с интервальными неопределенностями // Дифференциальные уравнения. 2021. Т. 57. № 7. С. 976—987. DOI: 10.31857/S0374064121070104.

13. **Makino K., Berz M.** Models and Their Applications // Numerical Software Verification 2017: conference. Heidelberg, Germany, July 22—23, 2017. Springer International Publishing AG, 2017. P. 3—13.

14. **Nataraj P. S. V., Sondur S.** The Extrapolated Taylor Model // Reliable Computing. July, 2011. P. 251—278.

15. **Рогалев А. Н.** Гарантированные методы решения систем обыкновенных дифференциальных уравнений на основе преобразования символьных формул // Вычислительные технологии. 2003. Том 8, № 5. С. 102—116.

16. **Fu C., Ren X., Yang Y.-F.** et al. Steady-state response analysis of cracked rotors with uncertain but bounded parameters using a polynomial surrogate method // Commun. Nonlinear Sci. Numer. Simul. 2019. Vol. 68. P. 240—256. DOI: 10.1016/j.cnsns.2018.08.004.

17. **Fu C., Xu Y., Yang Y.** et al. Response analysis of an accelerating unbalanced rotating system with both random and interval variables // J. Sound Vib. 2020. Vol. 466.Article 115047. DOI: 10.1016/j.jsv.2019.115047.

18. **Гидаспов В. Ю., Морозов А. Ю., Ревизников Д. Л.** Алгоритм адаптивной интерполяции с использованием TT-разложения для моделирования динамических систем с интервальными параметрами // Журнал вычислительной математики и математической физики. 2021. Том 61, № 9. С. 1416—1430. DOI: 10.31857/S0044466921090106.

19. **Смоляк С. А.** Квадратурные и интерполяционные формулы на тензорных произведениях некоторых классов функций // Докл. АН СССР, 1963. Том 148, № 5. С. 1042—1045.

20. **Bungatrz H-J., Griebel M.** Sparse grids // Acta Numerica. 2004. Vol. 13, No. 1. P. 147—269.

21. **Gerstner T., Griebel M.** Sparse grids // Encyclopedia of Quantitative Finance. 2008. Vol. 13. P. 5.

22. **Oseledets I. V.** Tensor-train decomposition // SIAM Journal on Scientific Computing. 2011. Vol. 33, No. 5. P. 2295—2317. DOI: 10.1137/090752286.

23. **Oseledets I., Tyrtyshnikov E.** TT-cross approximation for multidimensional arrays // Linear Algebra and its Applications. 2010. Vol. 432, Iss. 1. P. 70—88. DOI: 10.1016/j.laa.2009.07.024.

24. **Morozov A. Yu., Reviznikov D. L.** Modelling of Dynamic Systems with Interval Parameters on Graphic Processors // Про-

граммная инженерия. 2019. Том 10, № 2. С. 69—76. DOI: 10.17587/prin.10.69-76.

25. **Морозов А. Ю.** Параллельный алгоритм адаптивной интерполяции на основе разреженных сеток для моделирования динамических систем с интервальными параметрами // Программная инженерия. 2021. Том 12, № 8. С. 395—403. DOI: 10.17587/prin.12.395-403.

26. **Капралов Н. С., Морозов А. Ю., Никулин С. П.** Параллельная аппроксимация многомерных тензоров с использованием графических процессоров // Программная инженерия. 2022. Том 13, № 2. С. 94—101. DOI: 10.17587/prin.13.94-101.

27. **CUDA** Zone. URL: https://developer.nvidia.com/cuda-zone

28. **OpenMP.** URL: https://www.openmp.org/

29. **OpenMP** technology.URL: https://pvs-studio.com/ru/a/0057

30. **Kutta M.** Beitrag zur näherungsweisen Integration totaler Differentialgleichungen // Zeitschrift für Mathematik und Physik. 1901. Vol. 46. P. 435—453.

31. **Гилл Ф., Мюррей У., Райт М.** Практическая оптимизация. М.: Мир, 1985. 509 с.

32. **Kovacic I., Brennan M. J.** The Duffing Equation: Nonlinear Oscillators and their Behaviour. John Wiley & Sons, Hoboken, 2011. 369 p. DOI: 10.1002/9780470977859.

Parallel Algorithm for Parametric Identification of Dynamical Systems with Interval Parameters

A. Yu. Morozov, morozov@infway.ru,
Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences (FRC CSC RAS), Moscow, 119333, Russian Federation

Corresponding author:

Alexander Yu. Morozov, Researcher,
Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences (FRC CSC RAS), Moscow, 119333, Russian Federation
E-mail: morozov@infway.ru

Received on July 28, 2022
Accepted on August 10, 2022

The paper presents a parallel algorithm for the parametric identification of dynamical systems with interval parameters. The algorithm is based on the previously developed, substantiated and tested adaptive interpolation algorithm, which makes it possible to explicitly obtain the dependence of the states of a dynamic system on interval parameters. The solution of the problem of parametric identification is reduced to the problem of minimizing a certain objective function in the space of boundaries of interval parameter estimates. Due to the use of the adaptive interpolation algorithm when calculating the gradient of the objective function, there is no need for additional analysis and modeling of the original dynamic system, so it is convenient to use first-order methods for optimization. However, the task of calculating the objective function and the gradient includes a set of conditional minimization problems for explicit functions that can be solved independently of each other. The article discusses the main aspects and features of parallelization and implementation of the parametric identification algorithm and tests it on several representative examples. The acceleration and efficiency of parallelization are analyzed.

Keywords: *parallelization, OpenMP, parametric identification algorithm, adaptive interpolation algorithm, multidimensional interpolation, interval systems of ordinary differential equations, gradient methods, optimization*

For citation:

Morozov A. Yu. Parallel Algorithm for Parametric Identification of Dynamical Systems with Interval Parameters**,** *Programmnaya Ingeneria,* 2022, vol. 13, no. 10, pp. 497—507.

DOI: 10.17587/prin.13.497-507

References

1. **Nenarokomov A. V., Alifanov O. M., Krainova I. V.** et al. Estimation of environmental influence on spacecraft materials radiative properties by inverse problems technique, *Acta Astronautica*, 2019, vol. 160, pp. 323—330. DOI: 10.1016/j.actaastro.2019.04.014.

2. **Kabanikhin S. I., Kulikov I. M., Shishlenin M. A.** An Algorithm for Recovering the Characteristics of the Initial State of Supernova, *Computational Mathematics and Mathematical Physics.* 2020, vol. 60, pp. 1008—1016. DOI: 10.1134/S0965542520060135.

3. **Abgarian K. K., Noskov R. G., Reviznikov D. L.** The inverse coefficient problem of heat transfer in layered nanostructures, *Materials of Electronics Engineering.* 2017, vol. 20, no. 3, pp. 213—219. DOI: 10.17073/1609-3577-2017-3-213-219 (in Russian).

4. **Moore R. E., Kearfott R. B., Cloud M. J.** *Introduction to Interval Analysis,* SIAM, 2009, 223 p.

5. **Shary S. P.** *Finite dimensional interval analysis.* Novosibirsk, Institute of Computational Technologies SB RAS, XYZ Publisher, 2022, 653 p. (in Russian).

6. **Dobronets B. S.** *Interval Mathematics,* Krasnoyarsk State University, Krasnoyarsk, 2004, 216 p. (in Russian).

7. **Diligenskaya A. N., Samokish A. V.** Parametric identification in inverse heat conduction problems under conditions of interval uncertainty based on neural networks, *Bulletin of the Samara State Technical University*, 2020, vol. 28, no. 4 (68), pp. 6—18 (in Russian).

8. **Petrikevich Ya. I.** Structural-parametric identification of dynamic objects by interval initial data: dis. cand. tech. Sciences: 05.13.18. Kemerovo: Kemerovo State University, 2006. 225 p. (in Russian).

9. **Xiao N., Fedele F., Muhanna R. L.** Inverse Problems Under Uncertainties-An Interval Solution for the Beam Finite Element, *Conference Paper: 11th International Conference on Structural Safety & ReliabilityAt*, New York, NY, USA, 2013. 8 p. available at: https://www.researchgate.net/publication/269518192. DOI: 10.1201/b16387-430.

10. **Morozov A. Yu., Reviznikov D. L.** Interval approach to solving problems of parametric identification of dynamical systems, *Differential Equations*. 2022. Vol. 58, No. 7. pp. 962—976. DOI: 10.31857/S0374064122070 (in Russian).

11. **Morozov A. Yu., Reviznikov D. L., Gidaspov V. Yu.** Adaptive Interpolation Algorithm Based on a KD-Tree for the Problems of Chemical Kinetics with Interval Parameters, *Mathematical Models and Computer Simulations*, 2019, vol. 11, no. 4, pp. 622—633. DOI: 10.1134/S2070048219040100.

12. **Morozov A. Yu., Reviznikov D. L.** Adaptive Interpolation Algorithm on Sparse Meshes for Numerical Integration of Systems of Ordinary Differential Equations with Interval Uncertainties, *Differential Equations*, 2021, vol. 57, no. 7, pp. 947—958. DOI: 10.1134/S0012266121070107.

13. **Makino K., Berz M.** Models and Their Applications, *Numerical Software Verification 2017: conference,* Heidelberg, Germany, July 22—23, 2017, Springer International Publishing AG 2017, pp. 3—13.

14. **Nataraj P. S. V., Sondur S.** The Extrapolated Taylor Model, *Reliable Computing,* July, 2011, pp. 251—278.

15. **Rogalev A. N.** Guaranteed methods for solving systems of ordinary differential equations based on the transformation of symbolic formulas, *Computational technologies,* 2003, vol. 8, no. 5, pp. 102—116 (in Russian).

16. **Fu C., Ren X., Yang Y.-F., Lu K., Qin W.** Steady-state response analysis of cracked rotors with uncertain but bounded parameters using a polynomial surrogate method, *Commun. Nonlinear Sci. Numer. Simul.,* 2019, vol. 68, pp. 240—256. DOI: 10.1016/j.cnsns.2018.08.004.

17. **Fu C., Xu Y., Yang Y., Lu K., Gu F., Ball A.** Response analysis of an accelerating unbalanced rotating system with both random and interval variables, *J. Sound Vib.*, 2020, vol. 466, article 115047. DOI: 10.1016/j.jsv.2019.115047.

18. **Gidaspov V. Yu., Morozov A. Yu., Reviznikov D. L.** Adaptive Interpolation Algorithm Using TT-Decomposition for Modeling Dynamical Systems with Interval Parameters, *Computational Mathematics and Mathematical Physics*, 2021, vol. 61, no. 9, pp. 1387—1400. DOI: 10.1134/S0965542521090098.

19. **Smolyak S. A.** Quadrature and interpolation formulas for tensor products of certain classes of functions, *Dokl. Akad. Nauk SSSR*, 1963, vol. 148, no. 5, pp. 1042—1045 (in Russian).

20. **Bungatrz H-J., Griebel M.** Sparse grids, *Acta Numerica*, 2004, vol. 13, no. 1, pp. 147—269.

21. **Gerstner T., Griebel M.** Sparse grids, *Encyclopedia of Quantitative Finance,* 2008, vol. 13, p. 5.

22. **Oseledets I. V.** Tensor-train decomposition, *SIAM Journal on Scientific Computing*, 2011, vol. 33, no 5, pp. 2295—2317. DOI: 10.1137/090752286.

23. **Oseledets I., Tyrtyshnikov E.** TT-cross approximation for multidimensional arrays, *Linear Algebra and its Applications*, 2010, vol. 432, iss. 1, pp. 70—88. DOI: 10.1016/j.laa.2009.07.024.

24. **Morozov A. Yu., Reviznikov D. L.** Modelling of Dynamic Systems with Interval Parameters on Graphic Processors, *Programmnaya Ingeneria,* 2019, vol. 10, no. 2, pp. 69—76. DOI: 10.17587/prin.10.69-76.

25. **Morozov A. Yu.** Parallel Adaptive Interpolation Algorithm based on Sparse Grids for Modeling Dynamic Systems with Interval Parameters, *Programmnaya Ingeneria,* 2021, vol. 12, no. 8, pp. 395—403. DOI: 10.17587/prin.12.395-403.

26. **Kapralov N. S., Morozov A. Yu., Nikulin S. P.** Parallel Approximation of Multivariate Tensors using GPUs, *Programmnaya Ingeneria,* 2022, vol. 13, no. 2, pp. 94—101, DOI: 10.17587/prin.13.94-101.

27. **CUDA** Zone, available at: URL: https://developer.nvidia.com/cuda-zone

28. **OpenMP,** available at: URL: https://www.openmp.org/

29. **OpenMP** technology, available at: https://pvs-studio.com/ru/a/0057/

30. **Kutta M.** Beitrag zur näherungsweisen Integration totaler Differentialgleichungen. *Zeitschrift für Mathematik und Physik*, 1901, vol. 46, pp. 435—453.

31. **Gill P. E., Murray W., Wright M. H.** Practical Optimization, ACADEMIC PRESS, INC. San Diego, 1997, 509 p.

32. **Kovacic I., Brennan M. J.** *The Duffing Equation: Nonlinear Oscillators and their Behaviour,* John Wiley & Sons, Hoboken, 2011, 369 p. DOI: 10.1002/9780470977859.

ИНФОРМАЦИЯ

XVI Международная конференция инженерии программного обеспечения SECR/2023

17—18 марта 2023 г., Москва, Отель «Holiday Inn Сокольники»

SECR — одно из старейших и авторитетных ИТ-событий, посвященных индустрии разработки ПО.

Уникальность конференции — это широкий охват тем. SECR представляет разрез всей ИТ-индустрии и преподносит его участникам в виде 100+ различных докладов, мастер-классов и дискуссий за два дня. Современные подходы, тренды, прогнозы, экспертные мнения — все это SECR.

Программа SECR

- Выступления ключевых спикеров — приглашенных экспертов ИТ-рынка
- Доклады в несколько потоков, отобранные на конкурсной основе
- Дискуссии и круглые столы
- Полноценный трек мастер-классов
- Специальная SECR Party в конце первого дня.

Подробности: https://secrus.ru

DOI: 10.17587/prin.13.508-514

Designing a Conceptual Model of the Process of User Interface Construction*

S. A. Belikova, Senior Lecturer, belousova@sfedu.ru,
Y. I. Rogozov, Professor, Head of Department, yrogozov@sfedu.ru,
Southern Federal University, Taganrog, 347922, Russian Federation

Corresponding author:

Svetlana A. Belikova, Senior Lecturer,
Southern Federal University, Taganrog, 347922, Russian Federation
E-mail: belousova@sfedu.ru

Received on November 20, 2021
Accepted on September 30, 2022

Article proposes a conceptual model of the process of user interface construction for information systems, within which it is proposed to include the user into the design process, which will solve the existing problems of the adequacy of the interface and user activity in the subject area. In this case, the user is invited to compose a description of his activities in the subject area in a language close to natural by himself. The sequence of steps with the participation of the user is presented, which constitute a conceptual model of interface design. The scientific novelty of the research lies in a fundamentally different approach to the interface development, in which the user designs the application forms, relying primarily on his professional activity, and user doesn't need take into account the structure of the stored data or their processing functions, he connects data objects needed to his professional actions. The advantages of the proposed conceptual model are that it becomes possible to consider the user's activity as a whole, and on this basis to build an interface that best suits this activity, which will make the interface more understandable, reduce the level of discomfort when interacting with it, and increase user satisfaction.

Keywords: *user interface, user activity, mechanism of action, semantic approach, activity model, interface design*

For citation:

Belikova S. A., Rogozov Y. I. Designing a Conceptual Model of the Process of User Interface Construction, *Programmnaya Ingeneria,* 2022, vol. 13, no. 10, pp. 508—514.

УДК 004.5; 004.055; 004.4'2

С. А. Беликова, ст. препод., belousova@sfedu.ru, **Ю. И. Рогозов,** д-р техн. наук, профю, зав. каф., yrogozov@sfedu.ru,
Южный федеральный университет, Таганрог

Разработка концептуальной модели процесса конструирования пользовательского интерфейса

Предложена концептуальная модель процесса построения пользовательского интерфейса для информационных систем, в рамках которой предлагается включить пользователя в процесс проектирования, что позволит решить существующие проблемные вопросы адекватности интерфейса деятельности пользователя в предметной области. В таком случае пользователю предлагается самому составить описание своей деятельности, осуществляемой в предметной области, на языке, близком к естественному. Представлена последовательность шагов с участием пользователя, составляющих концептуальную модель проектирования

* The article was based on the materials of the report at the IX All-Russian Scientific Conference "Information Technologies for Intelligent Decision Support" ITIDS'2021

интерфейса. Научная новизна исследования заключается в принципиально ином подходе к разработке интерфейса, при котором пользователь проектирует формы приложений, опираясь в первую очередь на свою профессиональную деятельность. Пользователю не нужно учитывать структуру хранимых данных или функции их обработки, он связывает объекты данных, необходимые для его профессиональных действий. Преимущества предлагаемой концептуальной модели заключаются в возможности рассматривать деятельность пользователя в целом и на этой основе строить интерфейс, максимально соответствующий этой деятельности, что сделает интерфейс более понятным, снизит уровень дискомфорта при взаимодействии с ним и повысит удовлетворенность пользователей.

Ключевые слова: *пользовательский интерфейс, деятельность пользователя, механизм действия, смысловой подход, модель деятельности, проектирование интерфейса*

Исследование выполнено при финансовой поддержке РНФ в рамках научного проекта 22-19-00723.

Introduction

The characteristics of an effective interface of information systems (user productivity, minimal errors; high learning rate, subjective user satisfaction, etc.) depends on various aspects that were or were not taken into account in the interface design [1]. The existing problems in the field of user interface design can be summarized as follows:

- the interface is not adequate to the meaning of users activity (the structure of the user interface is developed in accordance with information flows (structure of objects) within the system itself, and not in accordance with the structure of the real users activity);
- the interface inadequately displays the objects of the system and the connections between them (the relative objects placement on the screen does not coincide with their logical connection or with their importance; redundancy of elements on forms);
- the complexity of the interface modification process;
- the interface is not adequate to the characteristics of users (the skills of using a computer and similar systems are partially solved due to the fact that the user himself takes part in the design and description of activities; the physical parameters of the user; subjective satisfaction when working with the interface);
- the interface is not adequate to the environment of use (for example, the presence of interruptions in user activities and so on).

On the other hand, development of complex information systems and user interfaces as its part requires the collaboration between team members with each other and between team members and users themselves. During the interface development process, we meet the difficulties connected with the correct understanding of the transmitted meaning from one team member to another and to the user at most. This situation arises because all design members are from heterogeneous domains, more precisely they think heterogeneously — within different concepts. So, the final interface can not fit the initial idea.

So user interface development process is usually organized by a life cycle model describing and guiding activities from the initial idea to the final implementation and performance testing, as for example the waterfall model. The problem with this approach is that it is required correct and complete understanding of the complete user interface design project from the beginning, as correcting a mistake made in a previous phase is a difficult and expensive task. Or similar situation, when requirements to the interface are changing during the system usage, is also difficult and expensive while changing the interface. So, it should be proposed some new approach that can help the customers and developers to spend less time and money to the process of interface correcting.

It is possible to try to solve these problems if we propose a way to transfer the structure and content of user activity in the subject area into the interface structure. In this regard, the existing approaches were considered that solve the listed problems in various ways, one of which is the involvement of a subject area expert in the development process [2].

A brief overview of existing approaches to the interface design

Currently, existing approaches tend to involve the user into the design process in the way of co-creation [3], however, different approaches implement user involvement in varying degrees. Many specialists and researchers notice the need for the direct participation of the user, since a software product is not the aim in itself — the goal is to satisfy the user's needs in simplifying his activities by means of a software system. In order to really take into account the needs of the user, it is necessary to constantly interact with him in the design process [4], or give him the opportunity to design the system and its interface by himself. In this regard,

such a direction as End-User Development (EUD) has arisen, which can be defined as a set of methods, technologies and tools that allow users of software systems to create or modify systems or their parts [5].

At the same time, the concept of user involvement in the development process has become an integral characteristic of any approach, only the involvement method differs: in some approaches, the user is engaged through conversation and interviews, while in others it is assumed that the user should do something according to the project by himself [6, 7]. The highest level of user involvement in the process of software systems development is observed in approaches where the user is provided with visual customization tools, after which he can get a finished working system or part of it [8].

Model-oriented approaches can involve user into the process of information systems co-creation in the best way [6, 7, 9, 10—13, 23]. The closest analogue is task-centered design [13]. This approach assumes that there is a detailed description of the user's tasks. Task Model allows to provide the structure and description of the tasks (actions) of the user that he can perform in the software system. Such models should reflect the content of user actions in the system: what should he do and why. There are two types of approach implementation, in the first version the tasks are represented by scripts in text form, in the second version hierarchical task trees are built [14]. The approach also assumes the possibility of assessing the developed interface by establishing a correspondence between the selected tasks and the interface components for whether all the functionality has been implemented [15, 16]. The disadvantage, from our point of view, is that a set of user tasks that he will perform using the designed system is considered, while the general outline of his professional activity is not considered at all, therefore it cannot be said that the problems of adequacy of tasks and their representations in the interface can be fully solved.

Task models are used not only in task-centered design approach but in other model-oriented approaches too. There are two types of task models [6]. The first type aims to reflect the sequence of tasks and their components, and the second — the data streams that are used when performing tasks [17]. But the point stays the same: the actions that the user performs in the subject area are not taken into account in this model, but only actions that directly relate to interaction with the system are considered. Other existing approaches were considered in [18—21].

There are some other types of models that are used in existing model-oriented approaches. The Dialog Model describes the structure of interaction between the user and the system: the structure of transitions from element to element depending on the action performed. As a model of dialogue, behavioral abstractions are usually used, for example, Petri nets, flowcharts, activity diagrams, UML state sequences [16].

The Application Model contains the structure of interaction between the system logic and the user interface, as well as the type of data transmitted during this interaction.

Domain Model is a correspondence of the concepts of the domain and the concepts of application logic and interface. Contains concepts, objects, operations, describing the subject area. The form of representation of such models are the essence of the subject area with the attributes, as well as possible relationships and operations on them.

The Presentation Model contains a high-level view of the interface, including what elements the user interface consists of and how these elements should be presented to the user.

The Behavior model has a similar context with the dialogue model. Describes how the user initiates a dialogue with the system, including a description of the input data, the controls used. The Control model contains a list of functions or operations that can be called, as well as preconditions and postconditions of their call. The Environment model contains the cultural aspect of the interaction context. Usually presented in the form of descriptions in natural language. User model most often contains a description of user characteristics, such as level of knowledge, physical and psychological qualities [6, 17, 23].

None of the above types of models contain explanations of the user's activity in the subject area, they are all at a lower level of abstraction and contain specific options for representing the interface and dialogue, therefore, they are focused on the developer, not on the user. In this case, the user may experience difficulties in checking the compliance of the functionality of the designed system, and the interfaces created in this way may not be sufficiently convenient, which will adversely affect the user performance (the consequences of a semantic gap problem [4, 21, 22, 24]).

As a result of considering the existing approaches to the user interface design, two reasons for the occurrence of the listed problems in the field of interface development can be distinguished:

- a semantic gap, which leads to the fact that the process of performing a task in the subject area is inadequate to the representation of this process in the user interface; a semantic gap arises in the process of communication between participants of the development process, when in the head of each participant there is a transformation of information received from another participant, the meaning of which may be distorted or partially lost;

• orientation of existing technologies for data processing, because most often the interface contains the purpose of data processing in the order in which the developer has determined.

To eliminate these reasons and achieve the preservation of the meaning of the user's actions in the subject area, and also fully reflect the user's requirements in the interface of the information system is possible if to create a common model which will be the same sign form of constructing various models: user activity; business processes at the same time; and it will be the initial data for system model. This universal model will be the result of the joint activity of all participants in the process of its creation, and, consequently, it will be understood and interpreted equally by all participants.

So we suggest approach to user interface development that can help end user and development team to correctly understand each other, and at the same time that can help to simplify the process of managing the interface structure if it is necessary.

Conceptual model of the interface construction process

The proposed approach [20, 25] aims to involve the end user as much as possible in the process of the interface design of the software system necessary for his needs. A user is understood as a competent problem domain specialist, i. e. a person with extensive knowledge and experience that allows him to effectively solve professional problems.

Within the framework of the proposed approach to interface design, the user is invited to compose a description of his professional activity in the subject area by himself.

The scientific novelty lies in a fundamentally different approach to the interface development, in which the user designs the application forms, relying primarily on his professional activity, and user doesn't need take into account the structure of the stored data or their processing functions, he connects data objects needed to his professional actions.

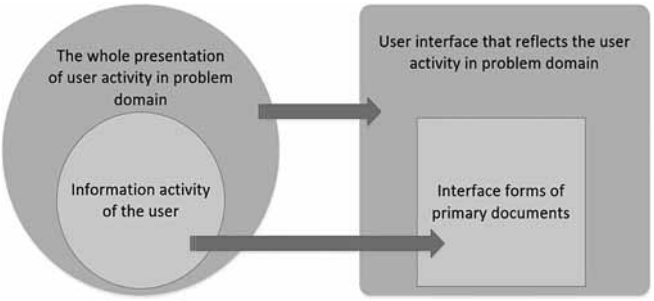


Figure 1. Concept of transformation of activities

As the initial data, a general description of the activity as a whole is singled out (Figure 1), and within the framework of this integral professional activity, informational activity is singled out, which is the activity that will be performed in the system and which is considered in existing approaches when constructing models.

Documents are the object of information activity, since the use of any information system comes down to drawing up, filling out documents based on the results of the activity. Therefore, the implementation of the forms of primary documents is necessary and should correspond to the information activity of the user. Information activity should be tied to professional activity. The interface created by the user should correspond to the structure of the user's professional activity in order for the user to find it convenient to navigate the system.

Conceptually, the model of the user interface design process consists of the following components (Figure 2).

• Description of the activity process in the subject area by the user in a language close to natural, in the form of a tree. Depending on the depth of detail, four levels are distinguished — actions, tasks, operations, steps.

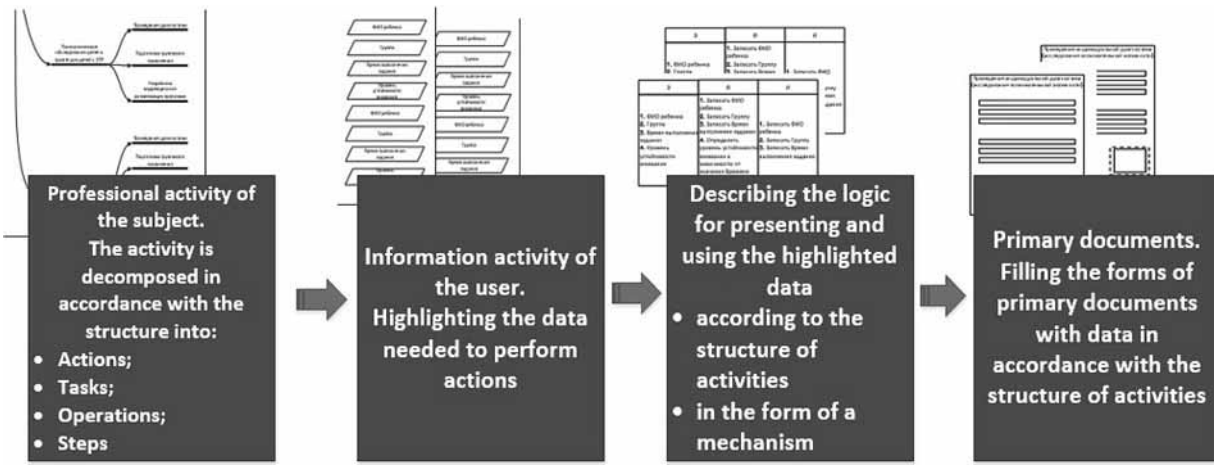


Figure 2. Conceptual model of the interface building process

- For each step of the user's activity, his information activity is singled out, which is the definition of data that are used and processed while the activity is performed.
- Transformation of the selected types of activity (professional and informational) into a model in the form of action mechanisms, in which the user's actions are decomposed to the level of previously prepared standard mechanisms for representing actions in the interface, i. e. there is a process of converting actions and data into document forms.
- Filling forms of documents with data in accordance with the structure of activities.

The interface will be a logically related set of interface forms filled with required controls. The communication logic of the interface forms is based on the connection of action mechanisms.

The process of user interface construction begins with the semantic analysis of the description of user activity made by user himself. It has to be determined the type of user action based on which the type of form is chosen.

Depending on the nature of the user's described activity in performing a certain professional function, a type of form is selected that can assist the user in performing this function. And the structure of this activity influences the filling of the selected interface form type with concrete elements.

So the initial data for the user interface design is the structure of the action mechanisms; the result is a set of related forms filled with related interface elements. Using a set of defined rules, information is extracted from the structure of the action mechanisms and the interface components are obtained.

An interface model which will be adequate to the user's professional activity can have a standard universal structure, which includes the following aspects:

- the level of the model of professional activity (allows the user to navigate in accordance with the state of information system and the stage of his work);
- the level of the information activity model (allows the user to navigate in accordance with the content of the screen forms of the stage of his work and its information results);
- the level of presentation of screen forms (allows the user to enter intermediate data necessary for the automated generation of primary documents);
- the level of presentation of the primary document (demonstrated optionally and allows the user to check the completeness and accuracy of the information entered);
- the level of description of the rules for converting intermediate data into fields of primary documents (demonstrated only at the editing stage and allows the user to describe the processes of filling in the fields of primary documents in subject area terms).

Development of basic modular architecture of platform for user interfaces construction

The first step in design of a platform for interfaces design by the user was the development of a basic architecture.

As part of the work on the platform basic architecture design, its modular structure was determined, consisting of four modules:

- the module of user activity description by the user himself;
- the module for describing the logic of presentation and use of selected data;
- the module for interface elements selection based on action mechanisms;
- the module for the user interface construction.

The module of user activity description by the user is necessary for the formation of user professional activity model. The input data for this module are the documents and functions of job descriptions that the users use in their professional activities. At the output of this module, data objects (information blocks related to the activity) and an activity model are formed.

The module describing the logic of presentation and use of selected data is necessary to describe the user professional actions as a structure of action mechanisms, in order to obtain informational activity. Information activity is a part of professional activity of the user that is performed with the help of information system. The input data for this module are data objects (informational blocks related to the activity). The output is the action mechanisms filled with concrete activity elements.

A mechanism-based interface element selection module is needed to correlate the underlying mechanisms for various interface elements and mechanisms describing logic and presentation and data usage. The inputs for this module are completed action mechanisms.

The input data for the user interface construction module are the model of activity, the documents with which user works, interface elements and action mechanisms for the formation and filling of the forms content. At the end, a ready-made interface is formed.

The developed basic modular architecture of the platform for user interfaces design is shown in Figure 3. The advantages of modular structure usage are presented in [26]. Based on the proposed approach there was developed a module of user interface construction by user himself [27]. The implementation of the module is demonstrated on the example of screens for adding actions to stages, forming an action diagram automatically, and binding functions. After making some actions in module the user gets the model of user interface for information system, which can be

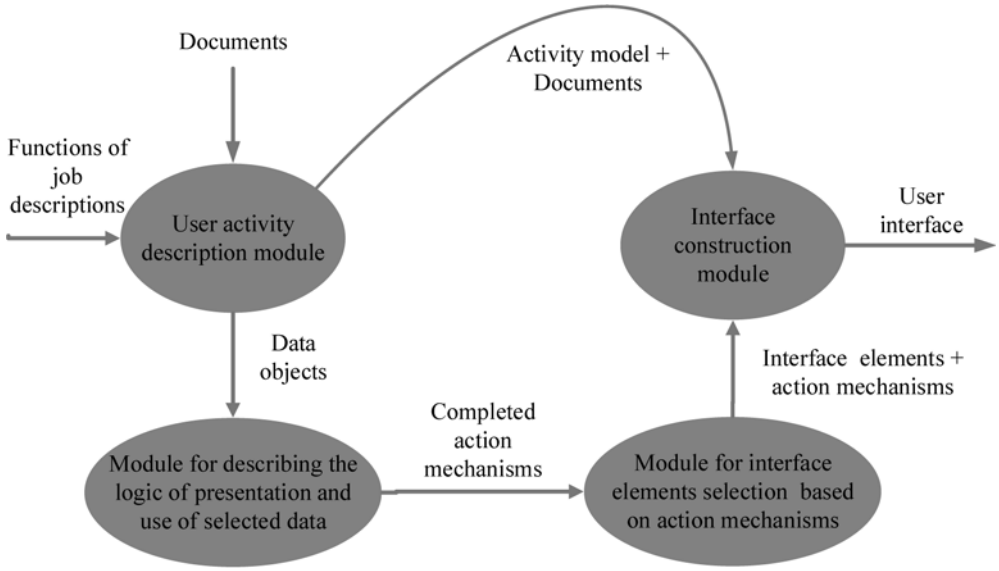


Figure 3. Basic modular architecture of a platform for user interfaces construction

transformed into program code manually, but automatization of this process is under considering.

Since the interface should contain data that is recorded in the process of business operations performing, there is often a problem that the relationship between interface elements is established based on the developers experience (including the basis of artificially formulated requirements for the interface), so usability problems arise, including the complexity of the relationship and redundancy of interface elements on the screen. The developed architecture of the platform solves this problem due to the fact that the connections of the interface elements are established on the basis of a user-defined structure of the task being performed, thereby the structure of the interface corresponds to the structure of the performed business operations so the logic of the business process performing is not broken.

In the developed architecture, due to the fact that the interface is an artificial component, there is a transition from the activity description (the result of the modeling stage) to the interface design by means of transformation: user actions in the domain are presented as they could be represented in the interface — with the addition of technical characteristics related to the structure of future forms, representation of objects in the form of interface objects, etc. The advantage is that it is based on the user's activity, which is presented in the form of actions in the interface. The above problem is solved by placing the empiricism into a strict framework of consideration, the system meaning is preserved in a single complex model. But at the same time, a new task arises — assessing the correspondence between the meaning displayed in the interface and the meaning inherent in the original task.

Conclusion

The idea of user interfaces design proposed in the article differs from the known ones by establishing and reflecting the set, structure and content of user actions within the subject area, while existing approaches do not fully take these aspects into account. This is achieved due to the fact that the source data will be a description of the end user tasks and functions for whom the designed interface is intended.

A holistic representation in the approach of the user professional activity brings the solution of listed problems of the semantic gap, since the use of the presented model allows us to consider the activity of the user as a whole, and on this basis to design an interface that most closely matches this activity, and also allows to reflect the subject area and the meaning of the user's activity in the system's interface. In addition, the approach corresponds to modern trends in involving the end user into the development process. The main quality of the proposed model that helps to manage the interface structure is that if we (or the user) somehow change the high-level description of actions presented with the mechanisms, we can get changing in the interface structure. Also the possibility of loss or distortion of semantic elements is eliminating which allows to design a more effective interface (more understandable for the user, including a decrease in the level of discomfort when interacting with it and an increase in overall user satisfaction).

Acknowledgments. *The study was financially supported by the Russian Science Foundation within the research project No. 22-19-00723.*

References

1. **Ngo D., Teo L., Byrne J.** Evaluating Interface Esthetics, *Knowledge and Information Systems*, 2002, vol. 4, pp. 46—79. DOI: 10.1007/s10115-002-8193-6.

2. **Gómez-Pérez J. M., Erdmann M., Greaves M. T., Corcho Ó.** A Formalism and Method for Representing and Reasoning with Process Models Authored by Subject Matter Experts, *IEEE Transactions on Knowledge and Data Engineering*, 2013, vol. 25, no. 9, pp. 1933-1945. DOI:10.1109/TKDE.2012.127.

3. **Rekrut M., Tröger J., Alexandersson J., Bieber D., Schwarz K.** Is Co-creation Superior to User Centred Design? Preliminary Results from User Interface Design for Inclusive Public Transport / J. Zhou, G. Salvendy (eds), *Aspects of IT for the Aged Population. Acceptance, Communication and Participation. ITAP 2018*, Springer, Cham., 2018, LNISA, vol. 10926, pp. 355—365. DOI: 10.1007/978-3-319-92034-4_27.

4. **Jia J., Capretz L. F.** Direct and mediating influences of user-developer perception gaps in requirements understa nding on user participation, *Requirements Engineering*, 2018, vol. 23, no. 2, pp. 277—290. DOI: 10.1007/s00766-017-0266-x.

5. **Paternò F.** End User Development: Survey of an Emerging Field for Empowering People, *International Scholarly Research Notices*, 2013, vol. 2013, article 532659. DOI: 10.1155/2013/532659.

6. **Trætteberg H.** *Model-based User Interface Design,* Norwegian University of Science and Technology, 2002, 211 p.

7. **Engel J., Herdin C., Märtin C.** Evaluation of Model-Based User Interface Development Approaches, *Human-Computer Interaction. Theories, Methods, and Tools, HCI 2014,* Springer, Cham., 2014, LNISA, vol. 8510, pp. 295—307. DOI: 10.1007/978-3-319-07233-3_28.

8. **Business** Process Management System, BPM. Upravleniye biznes-protsessami, rynok Rossii, available at: https://www.tadviser.ru/a/117491 (in Russian).

9. **Bertoni M., Bertoni A., Isaksson O. J.** EVOKE: A Value-Driven Concept Selection Method for Early System Design, *Journal of Systems Science and Systems Engineering,* 2018, vol. 27, pp. 46—77. DOI: 10.1007/s11518-016-5324-2.

10. **Ben Hassen M., Turki M., Gargouri F.** Towards Extending Business Process Modeling Formalisms with Information and Knowledge Dimensions, *Advances in Artificial Intelligence: From Theory to Practice, IEA/AIE 2017,* Springer, Cham., 2017, LNCS, vol. 10350, pp. 407—425. DOI: 10.1007/978-3-319-60042-0_45.

11. **Herre H.** General Formal Ontology (GFO): A Foundational Ontology for Conceptual Modelling, *Theory and Applications of Ontology: Computer Applications,* Dordrecht, Springer, 2010, pp. 297—345. DOI: 10.1007/978-90-481-8847-5_14.

12. **Kajiyama T., Satoh S.** An interaction model between human and system for intuitive graphical search interface, *Knowledge and Information Systems,* 2014, vol. 39, no. 1, pp. 41—60. DOI: 10.1007/s10115-012-0611-9.

13. **Lewis C. Rieman J.** Task-centered user interface design. A Practical Introduction. 2006, available at: http://www.hcibib.org/tcuid/

14. **Li J., Li-ying F., Qing X., Shi Z., Yiliu X.** Interface generation technology based on Concur Task Tree, 2010 *International Conference on Information, Networking and Automation (ICINA)*, IEEE, 2010, vol. 2, pp. 350—354. DOI: 10.1109/ICINA.2010.5636493.

15. **Yulius R., Neta F., Nasrullah M., Rambe M. F.** Implementation of Task-Centered Design in a Web-Based Catalogue, *Proceedings of the 2nd International Media Conference 2019 (IMC 2019),* 2020, pp. 381—386. DOI: 10.2991/assehr.k.200325.029.

16. **Greenberg S.** Working Through Task-Centered System Design, *Human-Computer Interact.*, 2004, pp. 49—66. DOI:10.11575/PRISM/30816.

17. **Crystal A., Ellington B.** Task analysis and human-computer interaction: approaches, techniques, and levels of analysis, *Proceedings of the Tenth Americas Conference on Information Systems,* New York, August 2004, 391 p.

18. **Belikova S., Rogozov Yu., Sviridov A., Lipko Ju.** Semantic technology of user interface development with the ability to reconfigure its structure, *19th International Multidisciplinary Scientific Geoconference SGEM 2019*, Sofia, 2019, pp. 579-586. DOI: 10.5593/sgem2019/2.1/S07.076.

19. **Belikova S. A., Rogozov Y. I., Sviridov A. S., Shevchenko O. V., Egorov A. V., Koltunova L. V.** Approach to user interfaces development based on semantic model of user activity, *Journal of Physics: Conference Series. 2019 12th International Conference on Computer and Electrical Engineering,* Institute of Physics Publishing, 2020, article 012012. DOI: 10.1088/1742-6596/1457/1/012012.

20. **Belikova S., Shevchenko O., Degtyareva E.** Integrative approach to the variable user interface development, *International Multidisciplinary Scientific GeoConference SGEM 2020*, Sofia, 2020, part 2.1, pp. 185—190. DOI: 10.5593/sgem2020/2.1/s07.024.

21. **Belikov A., Degtyarev A.** Development of a model for knowledge representation used at different life cycle stages of the information systems development, *International Multidisciplinary Scientific GeoConference SGEM 2020*, Sofia, 2020, part 2.1, pp. 111—117. DOI: 10.5593/sgem2020/2.1/s07.015.

22. **Abelein U., Paech B.** State of Practice of User-Developer Communication in Large-Scale IT Projects / C. Salinesi, I. van de Weerd (eds), *Requirements Engineering: Foundation for Software Quality REFSQ*, Springer, Cham, 2014, LNCS vol. 8396, pp. 95—111. DOI: 10.1007/978-3-319-05843-6_8.

23. **Vandervelpen C., Vandenbergh J., Luyten K., Coninx K.** Model-Based User Interface Development methods for the Design of Context-Driven, *Adaptable Interactive Systems*, 2004, D5.1, 27 p., available at: https: // distrinet.cs.kuleuven.be/projects/CoDAMoS/partners/usercommission/deliverables/deliverable5.1.pdf

24. **Jonas J. M.** Stakeholder integration in service innovation — a passive look from outside, *Stakeholder Integration in Service Innovation,* Springer Gabler, Wiesbaden, 2018, pp. 41—78.

25. **Belikov A., Sviridov A., Shevchenko O.** Analysis of the proposed semantic approach to design of information system interface, *17th International Multidisciplinary Scientific GeoConference SGEM 2017,* 2017, pp. 271—278. DOI: 10.5593/sgem2017/21/S07.035.

26. **Rogozov Y. I., Kucherov S. A., Lipko J. Y., Belikov A. N., Maakot A. Q., Belikova S. A.** Method of designing the modular structure of the information system, *Journal of Physics: Conference Series. 2019 12th International Conference on Computer and Electrical Engineering*, 2020, vol. 1457, article 012014. DOI:10.1088/1742-6596/1457/1/012014.

27. **Belikova S. A.** Development of user interface design module based on the use of integration-variable concept model, *Informatizaciya i svyaz*, 2022, no. 2, pp. 49—53. DOI: 10.34219/2078-8320-2022-13-2-49-53 (in Russian).

УДК 004.942
DOI: 10.17587/prin.13.515-524

В. И. Иордан, канд. физ.-мат. наук, доц., доц. кафедры[1], вед. математик[2], jordan@phys.asu.ru,
И. А. Шмаков[1], ст. преподаватель, ihammers.sia@gmail.com
[1] Федеральное государственное бюджетное образовательное учреждение высшего образования "Алтайский государственный университет", Барнаул
[2] Федеральное государственное бюджетное учреждение науки Институт теоретической и прикладной механики им. С. А. Христиановича СО РАН, Новосибирск

3D-визуализация гетерофазных интерметаллидных структур на основе компьютерного моделирования СВС в наноразмерной слоисто-блочной структуре Ti-Al

Приведены результаты термического и микроструктурного анализов самораспространяющегося высокотемпературного синтеза (СВС) интерметаллических соединений в наноразмерной атомной слоисто-блочной структуре Ti-Al, полученные с использованием известных программных пакетов LAMMPS и OVITO свободного доступа, а также с помощью программных средств, разработанных авторами статьи. С помощью пакета LAMMPS в версии параллельных вычислений, программная реализация которого основана на методе "молекулярной динамики", проводилось моделирование процесса СВС. Для 3D-визуализации результатов моделирования использовался пакет OVITO. Среди результатов микроструктурного анализа заслуживает внимания возникновение в ходе реакции СВС устойчивых гетерофазных интерметаллидных структур, подтвержденных с помощью пакета OVITO и созданного авторами программного модуля для расчета набора профилей плотности вещества СВС-продукта. В гетерофазных интерметаллидных структурах удалось обнаружить локальные особенности в виде слоистых квазикогерентных интерметаллических прослоек с наличием в них двумерных дефектов кристаллических решеток — плоскостей двойникования, разграничивающих интерметаллические прослойки на пары зерен (двойников) с симметричной переориентацией решеток.

Ключевые слова: *СВС, метод молекулярной динамики, система реагентов Ti-Al, элементарная кристаллическая ячейка, слоисто-блочная структура, температурный профиль, профиль плотности, гетерофазная интерметаллидная структура, дефект решетки, двойникование, параллельные вычисления, пакеты LAMMPS и OVITO*

Введение

Среди алюминидов титана в качестве перспективных конструкционных материалов можно выделить такие интерметаллидные сплавы, как $TiAl_3$, TiAl, Ti_3Al. Благодаря их прочностным свойствам, малой плотности, высокой коррозионной стойкости, жаростойкости и жаропрочности алюминиды титана оказались весьма привлекательными для различных областей машиностроения, в том числе авиационно-космической инженерии для создания защитных покрытий на технические изделия и материалы [1, 2].

В качестве эффективной технологии получения алюминидов титана уже несколько десятилетий (начиная с 1980-х гг.) используется самораспространяющийся высокотемпературный синтез (СВС), в процессе которого волна, возникающая за счет большого выделения тепловой энергии в результате экзотермической химической реакции компонентов смеси (например, тонкодисперсных порошков), распространяется по смеси от слоя к слою, в которых синтезируются твердые целевые продукты [3]. Воспламенение первого слоя можно добиться воздействием на поверхность смеси кратковременного теплового импульса, например, с помощью электрической спирали, лазерного луча и др.

Перед фронтом волны горения находится зона прогрева, формирующаяся теплопереносом.

За фронтом волны находится зона "догорания", в которой еще продолжаются химические реакции, и которая впоследствии определяет стадию структуро- и фазообразования с учетом вторичных физико-химических превращений, определяющих состав и структуру конечных продуктов СВС [3]. Свойства конечных продуктов во многом определяются влиянием начальных условий и следующих факторов: пористости смеси реагентов и их дисперсности; стехиометрического начального соотношения реагентов; степени разбавления смеси инертными добавками; температуры смеси; тепловых потерь во внешнюю среду; устойчивости движения фронта волны горения и др.

Поведение волны горения на макроскопическом уровне формируется процессами, происходящими в микрогетерогенной структуре волны горения, в которой проявляется пространственная дискретность горения в виде "микроочагов" СВС, влияющих на устойчивость горения.

Физические эксперименты для исследования микрокинетики СВС требуют применения дорогостоящего оборудования, дорогостоящих порошковых материалов в качестве реагентов смеси и продолжительного времени на их проведение. В дополнение к экспериментальным исследованиям в области СВС в последнее время стали эффективными и целенаправленными методы компьютерного моделирования процесса СВС и программные пакеты на их основе. Такие пакеты в конфигурации параллельных вычислений позволили ускорить выполнение большого числа вычислительных экспериментов (ВЭ) по сравнению с физическими экспериментами. В качестве примеров использования программных пакетов для моделирования микрокинетики СВС можно отметить известные, используемые в режиме свободного доступа пакеты LAMMPS [4], OVITO [5, 6].

Краткое описание функциональных возможностей программных пакетов LAMMPS и OVITO

В пакете LAMMPS программно реализован метод молекулярной динамики (МД), который в версии параллельных вычислений позволяет выполнять ВЭ по имитационному моделированию процесса СВС для атомных систем, содержащих до нескольких миллионов атомов.

В рамках метода МД (как метода статистической физики) у взаимодействующих между собой атомов изменяются микроскопические параметры (их координаты и скорости), а следовательно, в атомной системе изменяются макроскопические параметры: энергия, температура и давление. Равновесные состояния системы с некоторыми неизменными макропараметрами определяют понятия "канонических ансамблей" термодинамики в статистической механике.

Например, в каноническом ансамбле *NVT* и соответствующем ему термостате Носе—Гувера сохраняются следующие значения: число атомов *N*; объем *V*; температура *T*. Микроканонический ансамбль *NVE*, где *E* — внутренняя энергия системы, и соответствующий ему термостат Ланжевена можно использовать для быстрого достижения определенного равновесного состояния системы. Не требуя условия термостатирования, с помощью ансамбля *NVE* можно отслеживать эволюцию системы. Еще одним каноническим ансамблем является *NPT*-ансамбль с термостатом и баростатом, где *P* — давление.

Например, вычислительную процедуру метода МД можно определить с помощью многошагового алгоритма Верле [7, 8]:

$$\begin{cases} v_i(t+\Delta t/2) = v_i(t) + \dfrac{F_i(t)}{2m_i}\Delta t, \\ r_i(t+\Delta t) = r_i(t) + v_i(t+\Delta t/2)\Delta t, \\ F_i(t+\Delta t) = -\nabla_i U(r_i(t+\Delta t)), \\ v_i(t+\Delta t) = v_i(t+\Delta t/2) + \dfrac{F_i(t+\Delta t)}{2m_i}\Delta t, \end{cases} \quad (1)$$

в котором с учетом дискретного временного шага Δt и интегрированием уравнений движения (уравнений Ньютона) рассчитываются траектории частиц.

В системе (1): $v_i(t)$ — скорость i-го атома; $r_i(t)$ — радиус-вектор i-го атома; $F_i(t)$ — вектор-сила, действующая на i-й атом; m_i — масса i-го атома; $\nabla_i U(r_i(t))$ — градиент от потенциала $U(r_i(t))$, который определяет воздействующую на i-й атом силу $F_i(t)$. В первой и последней формулах системы уравнений (1) отношения значений сил $F_i(t)$ и $F_i(t+\Delta t)$ к массе m_i определяют значения ускорений i-го атома в моменты времени t и $t+\Delta t$ соответственно. Сила $F_i(t)$, действующая на i-й атом, определяется с помощью градиента от многочастичного потенциала $U(r_i(t))$ как суммарный результат взаимодействия с каждым "соседним" атомом (число соседних атомов определяется так называемым радиусом "обрезки"). Рассчитанные наборы координат и скоростей для всех атомов системы в последовательные моменты времени позволяют рассчитывать энергию системы атомов, температуру и другие параметры системы в те же моменты времени. Точность таких расчетов

зависит от выбора модельной функции потенциала межатомного взаимодействия $U(r)$.

Для исследования синтезируемых интерметаллических сплавов системы Ti-Al достаточно эффективным и универсальным потенциалом межатомного взаимодействия является модель "погруженного атома" (*embedded atom model* — EAM) [9]. Доступный для загрузки табличный файл EAM-потенциала (https://www.ctcms.nist.gov/potentials/Download/2003--Zope-R-R-Mishin-Y--Ti-Al/2/Zope-Ti-Al-2003.eam.alloy), предназначенный для системы Ti-Al, используется как входной файл для программного пакета LAMMPS (http://lammps.sandia.gov/). Использование EAM-потенциалов совместно с методом МД достаточно эффективно при моделировании и оценке механических, тепловых и структурных свойств металлических систем.

Пакет программ LAMMPS свободно распространяется по лицензии GPL и доступен в виде исходных кодов. Версия, написанная на C++ с интерфейсом передачи сообщений MPI, позволяет выполнять параллельные вычисления и значительно ускорить большой объем вычислений при симуляции исследуемых процессов [4]. Построение атомной системы, назначение потенциалов и настройки процедур моделирования реализуются в пакете LAMMPS с помощью входного конфигурационного файла (config.txt), предварительно создаваемого исследователем. Симуляция исследуемого процесса запускается последовательностью команд, записанных в файле config.txt.

Пакет LAMMPS использует модель "списков соседей" для частиц, которые отталкиваются на коротких расстояниях, поэтому локальная плотность частиц невысока, чем обеспечивается высокая вычислительная эффективность (процессоры обмениваются информацией о списках — о таких поддоменах).

Пакет OVITO [5, 6], в котором также используется пространственная декомпозиция на поддомены, применяется для 3D-визуализации исследуемых сложных объектов и процессов, а также для построения изображений с распределением атомов и типов структур, соответствующих элементарным ячейкам ГЦК (гранецентрированная кубическая), ОЦК (объемно-центрированная кубическая), ГПУ (гексагональная плотноупакованная) и т. д. (англ. аббр.: fcc, bcc, hcp и т. д.). Для рендеринга (отрисовки или визуализации) можно использовать способы трассировки лучей, реализованные в компонентах OpenGL, Tachyon, OSPRay и POV-Ray, которые можно включать в конфигурацию программы OVITO (либо отключать).

С учетом изложенного выше можно сказать, что важную роль в теоретических подходах к изучению процесса СВС играют программные пакеты и комплексы программ, создаваемые группами разработчиков. С одной стороны, такие разработчики глубоко погружены в предметную область СВС, с другой стороны, они профессионально владеют технологиями программирования и инструментальными средствами программной инженерии.

Цель исследования, результаты которого представлены в настоящей статье, — использование методологии применения и совершенствования комплекса программных средств вычислительной технологии. Такая технология призвана интегрировать в себе численные методы и методы компьютерного имитационного моделирования микрокинетики СВС в наноразмерных атомных слоисто-блочных структурах (СБС) системы Ti-Al с последующей компьютерной 3D-визуализацией процесса фазо- и структурообразования.

Постановка задач и подходы к их решению

Процесс создания комплекса программных средств для компьютерного имитационного моделирования микрокинетики СВС в наноразмерных атомных СБС системы Ti-Al с последующей компьютерной 3D-визуализацией процесса фазо- и структурообразования был разделен на перечисленные далее задачи (этапы), с которыми синхронизирована методика моделирования СВС.

Этап 1. *Создание (построение) исходной наноразмерной атомной СБС системы реагентов Ti-Al, характеризующейся шахматноподобным расположением чередующихся блоков.*

Для решения этой задачи используется пакет LAMMPS, на вход которого загружается упомянутый выше "конфигурационный" файл config.txt, предварительно создаваемый исследователем. В нескольких первых командных строках такого файла указываются: единицы измерения некоторых величин (массы, расстояния, времени, скорости, силы, энергии, температуры и т. п.); модель выбранного потенциала (например, EAM); условия поведения частиц (атомов) на границах системы (например, периодические краевые условия). В следующих строках файла config.txt задаются общие размеры исходной атомной системы и размеры каждого региона (слоя или блока). Кроме того, указывается схема расположения регионов и заполнения их кристаллическими ячейками атомов с указанием параметров ячеек (например, один блок с ГПУ-hcp-ячейками для атомов Ti, другой блок с ГЦК-fcc-ячейками для атомов Al).

Этап 2. *Моделирование процесса прогрева СБС однородно по всему объему структуры при постоянной начальной температуре T_0 в течение определенного непродолжительного времени для релаксации атомной структуры СБС.*

Моделирование релаксации СБС проводится в пакете LAMMPS запуском симуляции взаимодействия атомов в соответствии с алгоритмом МД последующими командами, записанными в файле config.txt. Такие команды формируются с учетом указанных параметров канонического *NPT*-ансамбля (в нашем случае — N = 1 028 175 атомов; давление P = 1 Бар; начальная температура T_0 = 800 K). Дополнительно указываются: длительность релаксации (в нашем случае — 0,4 нс) и параметр кинетической энергии для каждого атома системы, который должен вычисляться через указанный временной шаг (в нашем случае — через 0,1 нс) и далее на всех этапах моделирования. В конце каждого временного шага сохраняется на внешнем диске очередной выходной файл Res_i.dat (на данном этапе i = 1, 2, 3, 4). Такой файл содержит записи выходных данных по каждому из N атомов системы в следующем формате: тип атома (Ti или Al); координаты атома (x, y, z); скорости атома (v_x, v_y, v_z); действующие на атом силы (f_x, f_y, f_z); E_a — кинетическая энергия атома.

Этап 3. *Моделирование процесса инициирования (зажигания) СВС в наноразмерной атомной СБС (после ее релаксации).*

Аналогично этапу 2 в пакете LAMMPS продолжается симуляция процесса взаимодействия атомов в соответствии с алгоритмом МД следующими по списку файла config.txt командами. В этом списке указываются размеры начальной зоны — области атомной системы, которая подвержена линейному закону нагревания от 800 до 1400 K кусочно-ступенчатым способом. Это означает, что на каждом достаточно малом промежутке времени температура поднимается на одну "ступеньку", и в течение этого промежутка действуют условия *NVT*-ансамбля с новым значением температуры. Для оставшейся зоны атомной системы действуют условия *NVE*-ансамбля. Этап инициирования (зажигания) СВС длится 0,1 нс, и выходной файл Res_5.dat записывается на диск.

Этап 4. *Моделирование процесса СВС в среде Ti-Al в течение заданного времени моделирования.*

Аналогично этапу 3 в пакете LAMMPS продолжается симуляция процесса взаимодействия атомов в соответствии с алгоритмом МД следующими по списку файла config.txt командами, в которых указываются условия *NVE*-ансамбля для всей системы, а также периодические краевые условия для направлений Y и Z и свободные условия для направления X. С шагом по времени 0,1 нс на диск записываются очередные выходные файлы Res_i.dat, где i = 6, 7,....., K. Значение параметра K исследователь определяет в файле config.txt, и оно соответствует заранее планируемой продолжительности t_K моделируемого процесса СВС, где $t_K = K/10$ нс. На этом этапе волна горения трансформирует исходную СБС.

Этап 5. *Реализация 3D-визуализации результатов моделирования процесса СВС с помощью пакета OVITO и программных средств авторов статьи.*

Во-первых, в пакет OVITO последовательно загружаются выходные файлы Res_i.dat, выбранные исследователем для характерных моментов времени процесса СВС (им соответствует определенный набор индексов i). С использованием данных о типах всех атомов системы Ti-Al и их координат (x, y, z) формируется каждый раз 3D-изображение пространственной структуры распределения всех атомов в системе Ti-Al, либо можно получить 2D-изображение в заданном сечении объема системы Ti-Al. Кроме того, при открытии в интерфейсе вкладки Modify, раскрывается меню Pipeline editor (редактор конвейера обработки данных), в котором необходимо выбрать модификатор Ackland-Jones analysis. Тем самым можно получить 3D- или 2D-изображения распределения типов структур (ГЦК-fcc, ОЦК-bcc, ГПУ-hcp и т. д.) в объеме системы Ti-Al или в заданном сечении этого объема.

Во-вторых, для визуализации результатов моделирования СВС для каждого момента времени (с шагом 0,1 нс) в виде одномерных распределений температуры и плотности вещества вдоль направления волны горения (так называемых профилей температуры и плотности) авторами настоящей статьи ранее был разработан программный модуль T&P-profiles [11—13] на языке C/C++ (компилятор GCC). Идея алгоритма расчета профилей приведена далее.

Разбивая вдоль направления волны горения СВС с малым шагом 4 нм (т. е. вдоль оси X с ее предельным значением 420 нм) весь объем СБС на домены (параллелепипеды), в каждом из доменов оценивали осредненные температуры и плотности вещества. Длина домена 4 нм по оси X достаточно мала, а высота доменов (по оси Z) совпадает с высотой СБС и равна 16,5 нм. Глубина домена (по оси Y) совпадает с глубиной СБС и равна 2,5 нм (достаточно мала). Соответствующие размеры всех доменов одинаковы (4×2,5×16,5 нм) и их объем равен 165 нм^3. С использованием файлов Res_i.dat, для каждого домена с помощью циклов вычисляются по два счетчика: первый счетчик фиксирует число атомов первого типа (Ti), центры которых находятся внутри домена; второй счетчик — число атомов второго типа (Al), центры которых также находятся в этом домене. Умножая значения первого счетчика на массу атома Ti и значения второго счетчика на массу атома Al, а затем сумму этих результатов поделив на объем домена (165 нм^3), полу-

чаем значение плотности вещества домена. Аналогично, суммируя кинетические энергии тех атомов, которые входят в домен, рассчитываем суммарную внутреннюю энергию атомов домена, которая через константу Больцмана позволяет определять осредненную температуру в домене. Две последовательности значений температуры и плотности, соответствующие последовательности *X*-координат центров всех доменов, определяют понятия профилей температуры и плотности вещества СБС. Длина каждого из 12 блоков СБС равна 35 нм и на один блок приходится почти 9 доменов. Каждый из профилей содержит по 105 точек (отношение длины 420 нм к длине домена 4 нм).

Вследствие необходимости рассчитывать профили температуры и плотности для каждого момента времени (рассчитываем два семейства профилей), возникла потребность в ускорении расчетов. Для этого была использована утилита командной строки GNU parallel в качестве программной оболочки под управлением ОС GNU/Linux для параллельного выполнения однотипных заданий обработки блоков данных одинакового размера, но с различными значениями в них, для параллельного выполнения набора SIMD-задач (рис. 1).

В памяти вычислительного кластера создается необходимое число копий исполняемого файла T&P-profiles.exe, созданного с использованием GCC-компилятора языка программирования C/C++, и система GNU parallel с помощью загруженных в нее файлов Res_i.dat одновременно (параллельно) рассчитывает наборы профилей температуры и плотности.

Замечание: профили, приведенные ниже в данной статье, получены авторами с учетом предварительной коррекции алгоритмов расчета профилей и программного модуля T&P-profiles. Суть коррекции состоит в следующем.

В первоначальной версии алгоритмов расчета профилей два счетчика числа атомов двух типов (атомы первого типа — Ti, атомы второго типа — Al), входящих в анализируемый домен, допускали только целочисленные значения. Таким образом, если центр какого-либо атома, например, первого типа, оказывался в анализируемом домене, то только первый счетчик увеличивался на 1. Однако практически для каждого домена имеют место ситуации, когда некоторые атомы расположены своими частями одновременно в двух соседних (смежных) доменах. Это означает, что эти атомы разделены общей гранью (плоскостью) смежных доменов. Поэтому в таких случаях более корректно однотипные счетчики двух смежных доменов одновременно увеличивать на соответствующие им дробные доли объема атома (сумма долей равна 1). Таким образом, оба счетчика числа атомов двух типов, входящих в анализируемый домен, могут оказаться равными вещественным числам с дробной частью. Далее приведены результаты распределения числа атомов по доменам в тестовых примерах для регулярной кристаллической атомной решетки (например, для атомов Ti) при разбиении решетки на последовательность доменов с одинаковыми размерами.

- Для первоначальной версии алгоритмов расчета профилей общее целочисленное число атомов для каждого домена (сумма двух счетчиков для каждого домена), оказывается не постоянным числом, а флуктуация целочисленного числа атомов в доменах находится в диапазоне 0,5...1,2 %.
- Для скорректированного алгоритма, в котором значения счетчиков (следовательно, и их сумма) допускают вещественные значения с дробной частью, флуктуация нецелочисленного количества атомов в доменах менее 0,01 %, т. е. плотность вещества для атомной системы с регулярной решеткой оказывается практически постоянной. При этом при сложении суммы двух счетчиков по всем доменам получается практически целое число (отличие от целого менее 0,01 %), совпадающее с общим числом атомов исходной СБС системы Ti-Al.

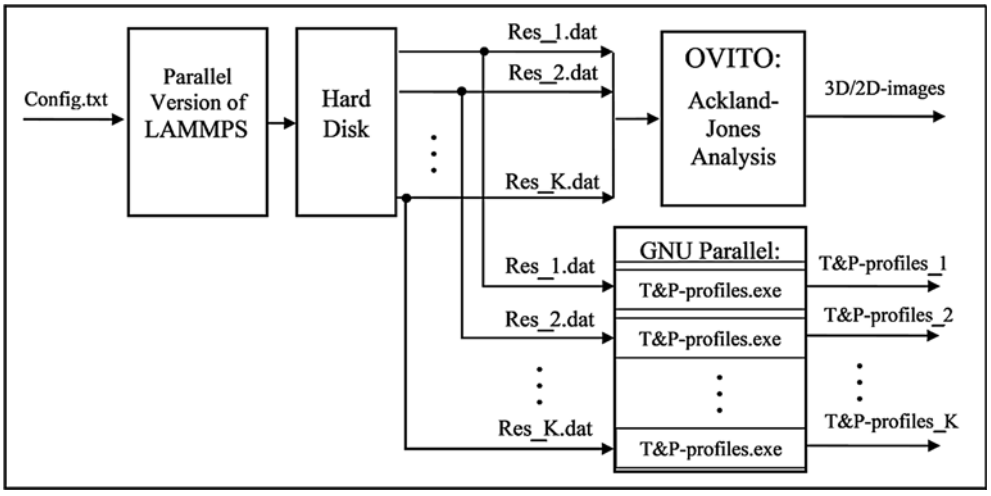


Рис. 1. Структурная схема комплекса программ для моделирования СВС

Методика и этапы молекулярно-динамического моделирования СВС в наноразмерных атомных СБС Ti-Al

В предыдущих публикациях авторы статьи исследовали микрокинетику СВС в наноразмерных атомных слоистых системах Ni-Al [10] и Ti-Al [11]. Структура слоистой системы Ti-Al (рис. 2, *а*, см. вторую сторону обложки) состоит из пяти крупных слоев, содержащих определенное число атомных плоскостей [11].

В слоистой системе Ni—Al при стехиометрии $N_{Ni}/N_{Al} = 3{,}94$ гетерофазные структуры не возникали [10].

Однако в слоистой системе Ti-Al со стехиометрией $N_{Ti}/N_{Al} = 1{,}23$ [11] гетерофазные интерметаллидные структуры были обнаружены (рис. 2, *б*, см. вторую сторону обложки). На последней временной диаграмме (маркер 16 ns, рис. 2, *б*, см. вторую сторону обложки), когда волна горения прошла почти до конца системы Ti-Al, отчетливо видны как минимум три гетерофазные структуры с резкими границами. Визуализация временных диаграмм (см. рис. 2, *б*, см. вторую сторону обложки) получена с помощью пакета OVITO.

В данной статье приведены результаты моделирования микрогетерогенного горения в атомной СБС с шахматноподобным чередованием наноразмерных блоков из атомов Ti и Al (рис. 3, см. вторую сторону обложки).

В каждом из пяти слоев находятся по 12 блоков длиной 35 нм. Верхний и нижний слои по высоте равны половине высоты каждого из трех внутренних слоев. Таким образом, высота внутреннего слоя равна приблизительно 4,125 нм, а третье измерение у блоков равно 2,5 нм. Другими словами, система Ti-Al в виде СБС состоит из 12 вертикальных штабелей по пять блоков. Размеры каждого штабеля определяются как 35×2,5×16,5 нм. В СБС входят 569 600 атомов Ti и 458 575 атомов Al, т. е. общее число атомов $N = 1\,028\,175$. Стехиометрия состава компонентов определяется отношением $N_{Ti}/N_{Al} = 1{,}242$, близким к стехиометрии 1,23 для слоистой структуры Ti-Al [11]. Доля атомов Ti равна 0,554 (55,4 %), доля атомов Al равна 0,446 (44,6 %). Ячейки атомов Ti соответствуют структуре гексагональной плотной упаковки (тип ячеек ГПУ-hcp). ГПУ-ячейки характеризуются параметрами: $a = 0{,}29508$ нм и $c = 0{,}46855$ нм, где a — длина ребра основания (правильного шестиугольника); c — высота ячейки, $c/a = 1{,}5879$ [9]. Атомы Al соответствуют структуре гранецентрированных кубических ячеек (тип ячеек ГЦК-fcc). ГЦК-ячейкам соответствует параметр $a = 0{,}405$ нм [10].

Методические условия по моделированию СВС в СБС (см. рис. 3) совпадают с условиями для слоистой структуры (см. рис. 2, *а*). Используется пакет LAMMPS с поддержкой параллельных вычислений и EAM-потенциал [9], методика моделирования предусматривает следующие этапы.

На этапе 1 весь объем СБС прогревался в течение 0,4 нс при температуре 800 K. При этом происходила релаксация атомной структуры с термодинамическими параметрами *NPT*-ансамбля $N = 1\,028\,175$, $P = 1$ Бар, $T = 800$ K. Кроме того, по всем трем измерениям атомной структуры сохранялись периодические граничные условия в течение 0,4 нс.

На этапе 2 моделирования в течение 0,1 нс в начальной зоне СБС в пределах 50 нм (как показано на рис. 3, см. вторую сторону обложки) проводилось линейное прогревание, начиная с 800 и до 1400 K, с сохранением периодических граничных условий для трех измерений. Прогревание в пределах 50 нм осуществлялось в условиях *NVT*-ансамбля (V — объем зоны прогревания), а в оставшейся части объема (в пределах 50...420 нм) в течение этих же 0,1 нс соблюдались условия *NVE*-ансамбля. По истечении 0,1 нс в исследуемой структуре фиксировались свободные граничные условия вдоль направления X, а по направлениям осей Y и Z сохранялись периодические условия. В результате этих действий в начальной зоне системы Ti-Al инициировался процесс СВС (процесс воспламенения). В последующее время (этап 3) при соблюдении условий *NVE*-ансамбля во всем объеме системы Ti-Al происходил процесс СВС с дальнейшим распространением волны горения вдоль направления оси X.

Термический анализ и интерпретация гетерофазных интерметаллидных структур в наноразмерной атомной СБС системы Ti-Al

Перейдем к анализу полученных профилей температуры (рис. 4, см. вторую сторону обложки). Каждый из профилей содержит по 105 точек, соответствующих доменам (отношение длины 420 нм к длине домена 4 нм).

Отношение линейного перемещения фронта волны горения ко времени этого перемещения (рис. 4, см. вторую сторону обложки) определяет оценку скорости движения фронта волны горения в СБС. Скорость движения фронта увеличивается приблизительно с 13 м/с (в пределах первых 12 нс) и до 22 м/с (от 12 до 24 нс). Таким образом, можно сделать вывод, что наблюдается режим разгона волны горения.

С момента воспламенения начальной зоны СБС наблюдаются сверхадиабатические температуры (почти до 2000 K) с последующей релаксацией до 1500 K (по времени около 5...6 нс). После 12 нс отчетливо видно, что в зонах прогрева, в которых еще отсутствует горение, начальная температура 800 K не сохраняется и начинает возрастать (до 1000 K), так как за предыдущее время в зону прогрева поступило существенное количество теплоты.

В конечной зоне прогревания системы Ti-Al к моменту времени 24 нс (и далее) температура очень быстро растет, поэтому зажигание СВС в этой зоне также сопровождается сверхадиабатическими температурами (до 1800 K). После релаксации сверхадиабатического выброса (к 32 нс) в конечной зоне устанавливается температура горения около 1600 K. Это на 100 K выше, чем температура, которая была практически неизменной для подавляющей части объема в предыдущие 24 нс. Более высокое значение 1600 K температуры горения в конечной зоне объясняется более высокой начальной температурой прогревания этой зоны (свыше 1000 вместо 800 K), что и послужило причиной заметного разгона движения фронта волны горения. К тому же, в СБС значение удельной поверхности контакта атомов Ti с атомами Al немного выше аналогичного параметра в слоистой структуре (рис. 2, *а*, см. вторую сторону обложки), т. е. в СБС диффузия атомов более интенсивна и скорость горения (следовательно, и скорость движения фронта волны горения) возрастает. Физические СВС-эксперименты, проводимые в тонких пленках системы Ni—Al (*nanofoils* [14]), подтверждают высокие скорости движения волны горения (до нескольких м/с).

Изображения, показанные на рис. 5 и 6 (см. третью сторону обложки), получены с помощью программы OVITO [5, 6].

Оценки скорости движения фронта волны горения, полученные на основе анализа изображений, представленных на рис. 5 и 6 (см. третью сторону обложки), практически совпадают с аналогичными оценками скорости фронта, полученными анализом температурных профилей (рис. 4, см. вторую сторону обложки). Кроме того, можно извлечь информацию о процессе взаимной диффузии атомов (рис. 5, см. третью сторону обложки) и о стабильном формировании гетерофазных интерметаллидных структур (рис. 6, см. третью сторону обложки).

На каждом из семи фрагментов рис. 5 видно, что в зонах "догорания" (после прохождения волны горения) СБС практически полностью разрушена (приблизительно к моменту времени 25 нс, см. рис. 5 и 6). Тем не менее, к моменту времени 32 нс на рис. 5 еще заметны локальные сосредоточения атомов Ti (однородного красного цвета).

На фрагментах рис. 6, начиная с 3-го и до последнего 7-го включительно, в тех же зонах, в которых на рис. 5 заметны локальные сосредоточения атомов Ti, отчетливо видны пять гетерофазных структур с резкими границами, внутри которых доминируют точки зеленого цвета и заметны узкие полоски-прожилки с доминированием в них точек красного цвета.

При определении типов элементарных ячеек с помощью пакета OVITO с использованием модификатора Ackland-Jones analysis (см. рис. 6) зеленый цвет точек соответствует ячейкам типа ГЦК-fcc, красный цвет точек — ячейкам типа ГПУ-hcp, синий цвет точек — ячейкам типа ОЦК-bcc.

С течением времени зоны с размещением блоков Ti и Al в СБС сокращаются и сохраняются лишь в тех местах, до которых волна горения еще не дошла. Блокам красного цвета (см. рис. 6) соответствуют блоки атомов Ti в состоянии фазы α-Ti (тип ячеек ГПУ-hcp), а блокам зеленого цвета — решетки атомов Al (тип ячеек ГЦК-fcc).

В пяти гетерофазных структурах (см. рис. 6) зеленым точкам (ячейкам ГЦК-fcc) не могут соответствовать скопления атомов Al, так как на рис. 5 в соответствующих им местах отсутствуют зоны с доминированием зеленого цвета, соответствующего атомам Al (т. е. на рис. 5 зеленые точки рассеяны равномерно так же, как и вне пяти гетерофазных структур). Необходимо отметить, что ячейкам типа ГЦК-fcc структурно близки гранецентрированные тетрагонально-искаженные ячейки (обозначим их как ячейки типа ГЦТ-fct), соответствующие интерметаллической γ-фазе TiAl. По этой причине в пяти гетерофазных структурах доминирует γ-фаза TiAl (см. рис. 6 — зеленые точки в гетерофазных структурах). Узким прожилкам темно-красного цвета (тип ячеек ГПУ-hcp) в пяти гетерофазных структурах соответствуют сосредоточения фазы α-Ti. Сопоставляя между собой температурные профили (см. рис. 4) с образующимися гетерофазными структурами (см. рис. 6) для одного и того же момента времени, можно видеть, что гетерофазные структуры формируются при температуре, превышающей 1400 K, т. е. превышающей температуру перитектического распада интерметаллида Ti_3Al. Другими словами, для стехиометрии $N_{Ti}/N_{Al} = 1{,}242$ в предвоспламенительной зоне (см. рис. 4, температуры 1100...1400 K) согласно равновесной диаграмме состояния для системы Ti-Al [15] формируется двухфазная область Ti_3Al + TiAl, и с повышением температуры горения (свыше 1400 K) фаза Ti_3Al перитектически распадается и нарастает количество фазы TiAl.

В подтверждение анализа, выполненного на основе пакета OVITO, можно указать на выводы, которые вытекают из анализа равновесной диаграммы состояния для системы Ti-Al [15]: при температурах 1470...1600 K (см. рис. 4, маркер 32 нс) для стехиометрии $N_{Ti}/N_{Al} = 1{,}242$ в системе Ti-Al [15] должны формироваться две двухфазные области: α-Ti + + TiAl (при температурах меньше 1513 K) и β-Ti + + TiAl (при температурах свыше 1513 K). Фазе β-Ti соответствуют ячейки типа ОЦК-bcc (им на рис. 6 соответствуют синие точки, которые частично наблюдаются и в пяти гетерофазных структурах). Вне гетерофазных структур точек синего цвета еще больше, поэтому можно утверждать о присутствии фазы β-Ti в определенной степени по всему объему реагирующей системы.

Как видно на рис. 2, *б* (для слоистой системы Ti-Al) и рис. 6 (для СБС Ti-Al), вне гетерофазных структур в обоих случаях наблюдается смесь различных цветов (красный, синий, зеленый, желтый и белый), соответствующая неупорядоченной смеси различных структур и интерметаллических фаз (кроме фаз TiAl, α-Ti, β-Ti и Ti_3Al присутствует и фаза $TiAl_3$). Значения плотностей фаз α-Ti, Ti_3Al и TiAl существенно выше значений плотности фазы $TiAl_3$ и смеси вещества, распределенного вне гетерофазных структур. Этот факт, который следует из анализа рис. 5 и 6 (на основе пакета OVITO), подтверждается другим методом анализа, основанного на построении профилей плотности вещества (рис. 7, см. третью сторону обложки) вдоль направления распространения волны горения СВС (оси *X*).

Метод расчета и анализа профилей плотности был апробирован авторами ранее в работе [11] и в последующих публикациях, включая [12, 13].

После прохождения волны горения через СБС (после 25 нс), на последнем профиле плотности (маркеры 32 нс, см. рис. 7) наблюдаются пять пиков плотности, соответствующих расположению гетерофазных структур на рис. 6. Кроме того, эффективная ширина пиков плотности (около 25 нм) практически совпадает с шириной гетерофазных интерметаллидных структур (согласно рис. 6), т. е. согласуются оценки размеров гетерофазных структур, полученных двумя разными методами (визуализация OVITO и профили плотности).

Рисунки 5, 6 и 7 (см. третью сторону обложки) главным образом способствовали анализу процесса структурообразования в реагирующей системе Ti-Al. Однако после 25 нс прореагировавшая система Ti-Al переходит в состояние так называемой зоны догорания и эволюция ее структуры и фазообразования на этом не заканчивается.

Отметим, что в реальном процессе СВС состояние догорания по времени длится достаточно долго (в пределах нескольких десятков секунд, а не наносекунд), продолжаясь даже и при остывании СВС-системы. Это означает, что на второй стадии СВС продолжаются вторичные физико-химические превращения, определяющие состав и структуру конечных продуктов СВС (интерметаллидов титана).

В нашем случае процесс моделирования вторую стадию процессов СВС не охватывает. Причина в том, что в таком случае затрачиваемое на моделирование второй стадии время оказалось бы в миллиард раз больше, чем затраченные нескольких суток на моделирование первой стадии процесса СВС (с модельным временем 32 нс). Далее продолжим анализ локальных особенностей в образованных гетерофазных интерметаллидных структурах (рис. 8 и 9, см. четвертую сторону обложки).

Возникновение гетерофазных интерметаллидных структур (см. рис. 6) и подструктур (см. рис. 8 и 9) внутри этих гетерофазных структур в условиях температурно-силовых воздействий на атомы связано с проявлением всплесков давления, внутренних дефектов и движением дислокаций в локальных структурах атомной системы Ti-Al.

На рис. 8 (см. четвертую сторону обложки) показаны 3D-снимки (в увеличенном масштабе) двух различных распределений атомов для одной и той же гетерофазной структуры, которая отражена первой (слева) на последнем фрагменте рис. 6 (маркер 32 нс, см. третью сторону обложки).

На рис. 9 (см. четвертую сторону обложки) отражены нижние половины этих двух 3D-снимков рис. 8 с дополнительным увеличением масштаба визуализации. Сопоставляя полосу с доминированием окрашенных в красный цвет атомов (рис. 9, *б*) с соответствующей ей полосой фрагмента на рис. 9, *а*, видно, что в правой части имеется монолитный блок из атомов Ti на фрагменте рис. 9, *а*, так же, как и на фрагменте рис. 9, *б* окрашенных в красный цвет с принадлежностью к структурам ГПУ-hcp типа. Монолитный блок атомов Ti соответствует фазе α-Ti. Левее него в этой же полосе согласно фрагменту рис. 9, *а* явно выражен результат растворения блока атомов Ti атомами Al. Согласно фрагменту на рис. 9, *б* здесь с учетом доминирования красного цвета возможно только одно интерметаллическое соединение со структурами типа ГПУ-hcp — это фаза Ti_3Al.

Кроме того, в монолитном блоке фазы α-Ti (см. рис. 9, *а*) наблюдается двумерный дефект — вертикальная плоскость двойникования, перпендикулярная к плоскости снимка, разделяющая два зерна (двух двойников) с симметричной пространственной переориентацией их кристаллических решеток. Атомы, помеченные синим цветом на рис. 9, *б*, как уже отмечалось выше, соответствуют типу ячеек ОЦК-bcc и указывают на рассеянное распределение фазы β-Ti в гетерофазной интерметаллидной структуре (ГФИС). Скопления же атомов, помеченных зеленым цветом (см. рис. 9, *б*), на что обращалось внимание ранее, соответствуют типу ячеек ГЦК-fcc (и указывают на доминирование фазы TiAl в ГФИС.

Заключение

Продемонстрирована целесообразность применения программных пакетов LAMMPS и OVITO для МД-моделирования и визуализации процесса СВС в наноразмерной СБС с шахматноподобным расположением блоков из атомов Ti и Al. Показана эффективность разработанного авторами программного модуля для расчета профилей температуры

и плотности вещества, позволивших осуществлять температурный анализ микрокинетики СВС и распознавать образующиеся интерметаллические фазы.

Вычислительные эксперименты с использованием метода МД-моделирования СВС позволили обнаружить в процессе фазо- и структурообразования гетерофазные интерметаллидные структуры, связанные с проявлением всплесков давления, внутренних дефектов и движением дислокаций в условиях температурно-силовых воздействий на атомные блоки системы Ti-Al. Внутри гетерофазных интерметаллидных структур в ГПУ-прослойках обнаружен двумерный дефект — плоскость двойникования, разделяющая два зерна (двух двойников) с различающейся пространственной ориентацией их решеток.

Параллельные вычисления с использованием пакета LAMMPS в версии языка C++ совместно с интерфейсом MPI выполнялись на кластере ограниченной вычислительной мощности из 15 ПК (с четырехядерными процессорами) с возможностью распараллеливать вычисления в каждом вычислительном эксперименте на 60 параллельных потоков для атомной системы Ti-Al с достаточно большим общим числом атомов (1 028 175 атомов) по отношению к вычислительной мощности использованного кластера. При этих условиях была достигнута следующая оценка производительности моделирования процесса СВС: за сутки непрерывного времени расчетов на кластере из 15 ПК моделировался процесс СВС длительностью почти 2 нс реального времени.

Список литературы

1. **Пячин С. А., Ершова Т. Б., Бурков А. А., Власова Н. М., Комарова В. С.** Использование алюминидов титана для создания электроискровых покрытий // Известия вузов. Порошковая металлургия и функциональные покрытия. 2015. № 1. С. 55—61. DOI: 10.17073/1997-308X-2015-1-55-61.
2. **Аванесян Т. Ч.** Особенности высокотемпературного окисления и микродугового оксидирования сплавов на основе γ-TiAl: дис. ... канд. хим. наук.: 05.17.03. М., 2014. 159 с.
3. **Мержанов А. Г.** Твердопламенное горение. Черноголовка: ИСМАН, 2000. 224 с.
4. **Plimpton S.** Fast Parallel Algorithms for Short-Range Molecular Dynamics // J. Comp. Phys. 1995. No. 117. P. 1—19. DOI: 10.1006/jcph.1995.1039.
5. **Stukowski A.** Visualization and analysis of atomistic simulation data with OVITO — the Open Visualization Tool // Modelling and Simulation in Materials Science and Engineering. 2010. Vol. 18, Iss. 1. Article ID 015012. DOI: 10.1088/0965-0393/18/1/015012.
6. **Ackland G. J., Jones A. P.** Applications of local crystal structure measures in experiment and simulation // Phys. Rev. B. 2006. Vol. 73, Iss. 5. Article ID 054104. DOI: 10.1103/PhysRevB.73.054104.
7. **Verlet L.** Computer "experiments" on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules // Phys. Rev. 1967. Vol. 159, No. 1. P. 98—103. DOI: 10.1103/PhysRev.159.98.
8. **Verlet L.** Computer "experiments" on classical fluids. II. Equilibrium correlation functions // Phys. Rev. 1968. Vol. 165, No. 1. P. 201—214. DOI: 10.1103/PhysRev.165.201.
9. **Zope R. R., Mishin Y.** Interatomic Potentials for Atomistic Simulations of the Ti-Al System // Phys. Rev. B. 2003. Vol. 68, Iss. 2. Article ID 024102. DOI: 10.1103/PhysRevB.68.024102.
10. **Шмаков И. А., Иордан В. И., Соколова И. Е.** Компьютерное моделирование СВ-синтеза алюминида никеля методом молекулярной динамики в пакете LAMMPS с использованием параллельных вычислений // Высокопроизводительные вычислительные системы и технологии. 2018. Том 2, № 1. С. 48—54.
11. **Jordan V. I., Shmakov I. A.** Reproducibility of a heterophase structure emergence effect when changing the ignition temperature of SHS in a layered nanosized nonstoichiometric Ti-Al system // IOP Conf. Journal of Physics: Conf. Series, 2019. Vol. 1281. Article ID 012030. DOI: 10.1088/1742-6596/1281/1/012030.
12. **Jordan V. I., Shmakov I. A.** Thermal and microstructural analysis of intermetallide synthesis in the Ni-Al layered-block atomic structure based on the computer-aided simulation of SHS // Communications in Computer and Information Science. 2020. Vol. 1304. P. 43—61. DOI: 10.1007/978-3-030-66895-2_4.
13. **Jordan V. I., Shmakov I. A.** Method for Intermetallide Spatial 3D-Distribution Recognition in the Cubic Ni@Al "Core-Shell" Nanoparticle based on Computer MD-Simulation of SHS // Communications in Computer and Information Science. 2022. Vol. 1526. P. 101—120. DOI: 10.1007/978-3-030-94141-3_9.
14. **Turlo V., Politano O., Baras F.** Microstructure evolution and self-propagating reactions in Ni—Al nanofoils: an atomic-scale description // J. Alloys and Compd. 2017. Vol. 708. P. 989—998. DOI: 10.1016/j.jallcom.2017.03.051.
15. **Григоренко Г. М., Костин В. А., Григоренко С. Г.** Расчет равновесных диаграмм состояния и фазовых превращений титановых сплавов системы титан—алюминий // Современная электрометаллургия. 2018. № 3. С. 39—44. DOI: 10.15407/sem2018.03.06.

3D Visualization of Heterophase Intermetallic Structures Based on Computer Simulation of SHS in a Nanosized Ti-Al Layered-Block Structure

V. I. Jordan, PhD, Associate Professor[1], Leading Mathematician[2], jordan@phys.asu.ru,
I. A. Shmakov, Senior Lecturer[1], ihammers.sia@gmail.com,

Corresponding author:

Vladimir I. Jordan, PhD, Associate Professor[1], Leading Mathematician[2], jordan@phys.asu.ru
[1] Altai State University, Barnaul, 656049, Russian Federation
[2] Khristianovich Institute of Theoretical and Applied Mechanics, SB RAS, Novosibirsk, 630090, Russian Federation

Received on July 27, 2022
Accepted on September 22, 2022

The paper presents the results of thermal and microstructural analysis of the SHS of intermetallic compounds in a nanosized atomic layer-block structure of Ti-Al, obtained using the freely available software packages LAMMPS and OVITO, as well as using the software of the authors of the paper. Using the LAMMPS package in the version of parallel computing, the software implementation of which is based on the "molecular dynamics" method, the SHS process is simulated. For 3D visualization of simulation results, the OVITO package is used. Among the results of microstructural analysis, noteworthy is the appearance of stable heterophase intermetallic structures during the SHS reaction, which were confirmed using the OVITO package and the software module developed by the authors for calculating a set of matter density profiles of the SHS-product. In heterophase intermetallic structures, it was possible to detect local features in the form of layered quasi-coherent intermetallic interlayers with the presence of two-dimensional defects in crystal lattices — twinning planes delimiting pairs of grains (twins) with symmetrical lattices reorientation.

Keywords: *SH-synthesis, molecular dynamics method, Ti-Al reagent system, unit cell, layered block structure (LBS), temperature profile, density profile, heterophase intermetallic structure, lattice defect, twinning, parallel computing, LAMMPS and OVITO packages*

For citation:

Jordan V. I., **Shmakov I. A.** 3D Visualization of Heterophase Intermetallic Structures Based on Computer Simulation of SHS in a Nanosized Ti-Al Layered-Block Structure, *Programmnaya Ingeneria,* 2022, vol. 13, no. 10, pp. 515—524

DOI: 10.17587/prin.13.515-524

References

1. **Pyachin S. A., Ershova T. B., Burkov A. A., Vlasova N. M., Komarova V. S.** Using titanium aluminides to create electrospark coatings, *Izvestiya vuzov. Powder metallurgy and functional coatings*, 2015, no. 1, pp. 55—61. DOI: 10.17073/1997-308X-2015-1-55-61 (in Russian).

2. **Avanesyan T. Ch.** Features of high-temperature oxidization and microarc oxidation of alloys based on γ-TiAl, Diss. ... Ph.D.: 05.17.03. Moscow, 2014, 159 p. (in Russian).

3. **Merzhanov A. G.** *Solid-Flame Combustion*, Chernogolovka, Izd. ISMAN Publisher, 2000, 224 p. (in Russian).

4. **Plimpton S.** Fast Parallel Algorithms for Short-Range Molecular Dynamics, *J. Comp. Phys.*, 1995, no. 117, pp. 1—19. DOI: 10.1006/jcph.1995.1039.

5. **Stukowski A.** Visualization and analysis of atomistic simulation data with OVITO — the Open Visualization Tool, *Modelling and Simulation in Materials Science and Engineering*, 2010, vol. 18, iss. 1, article ID 015012. DOI: 10.1088/0965-0393/18/1/015012.

6. **Ackland G. J., Jones A. P.** Applications of local crystal structure measures in experiment and simulation, *Phys. Rev. B.,* 2006, vol. 73, iss. 5, article ID 054104. DOI: 10.1103/PhysRevB.73.054104.

7. **Verlet L.** Computer "experiments" on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules, *Phys. Rev.,* 1967, vol. 159, no. 1, pp. 98—103. DOI: 10.1103/PhysRev.159.98.

8. **Verlet L.** Computer "experiments" on classical fluids. II. Equilibrium correlation functions, *Phys. Rev.,* 1968, vol. 165, no. 1, pp. 201—214. DOI: 10.1103/PhysRev.165.201.

9. **Zope R. R., Mishin Y.** Interatomic Potentials for Atomistic Simulations of the Ti-Al System, *Phys. Rev. B.,* 2003, vol. 68, iss. 2, article ID 024102. DOI: 10.1103/PhysRevB.68.024102.

10. **Shmakov I. A., Jordan V. I., Sokolova I. E**. Computer simulation of the SH-synthesis of nickel aluminide by the molecular dynamics method in the LAMMPS package using parallel computing, *High-performance computing systems and technologies,* 2018, vol. 2, no. 1, pp. 48—54 (in Russian).

11. **Jordan V. I., Shmakov I. A.** Reproducibility of a heterophase structure emergence effect when changing the ignition temperature of SHS in a layered nanosized nonstoichiometric Ti-Al system, *IOP Conf. Journal of Physics: Conf. Series,* 2019, vol. 1281, article ID 012030. DOI: 10.1088/1742-6596/1281/1/012030.

12. **Jordan V. I., Shmakov I. A.** Thermal and microstructural analysis of intermetallide synthesis in the Ni-Al layered-block atomic structure based on the computer-aided simulation of SHS, *Communications in Computer and Information Science,* 2020, vol. 1304, pp. 43—61. DOI: 10.1007/978-3-030-66895-2_4.

13. **Jordan V. I., Shmakov I. A.** Method for Intermetallide Spatial 3D-Distribution Recognition in the Cubic Ni-Al "Core-Shell" Nanoparticle based on Computer MD-Simulation of SHS, *Communications in Computer and Information Science,* 2022, vol. 1526, pp. 101—120. DOI: 10.1007/978-3-030-94141-3_9.

14. **Turlo V., Politano O., Baras F.** Microstructure evolution and self-propagating reactions in Ni-Al nanofoils: an atomic-scale description, *J. Alloys and Compd.,* 2017, vol. 708, pp. 989—998. DOI: 10.1016/j.jallcom.2017.03.051.

15. **Grigorenko G. M., Kostin V. A., Grigorenko S. G.** Calculation of equilibrium state diagrams and phase transformations of titanium alloys of the titanium-aluminum system, *Modern Electrometallurgy,* 2018, no. 3, pp. 39—44. DOI: 10.15407/sem2018.03.06 (in Russian).

ООО "Издательство "Новые технологии". 107076, Москва, ул. Матросская Тишина, д. 23, стр. 2
Технический редактор *Е. М. Патрушева*. Корректор *А. В. Чугунова*.

Сдано в набор 14.10.2022 г. Подписано в печать 21.11.2022 г. Формат 60×88 1/8. Заказ PI1021
Цена свободная.

Оригинал-макет ООО "Адвансед солюшнз". Отпечатано в ООО "Адвансед солюшнз".
119071, г. Москва, Ленинский пр-т, д. 19, стр. 1. Сайт: **www.aov.ru**

Рисунки к статье **В. И. Иордана, И. А. Шмакова**

«3D-ВИЗУАЛИЗАЦИЯ ГЕТЕРОФАЗНЫХ ИНТЕРМЕТАЛЛИДНЫХ СТРУКТУР НА ОСНОВЕ КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ СВС В НАНОРАЗМЕРНОЙ СЛОИСТО-БЛОЧНОЙ СТРУКТУРЕ Ti-Al»

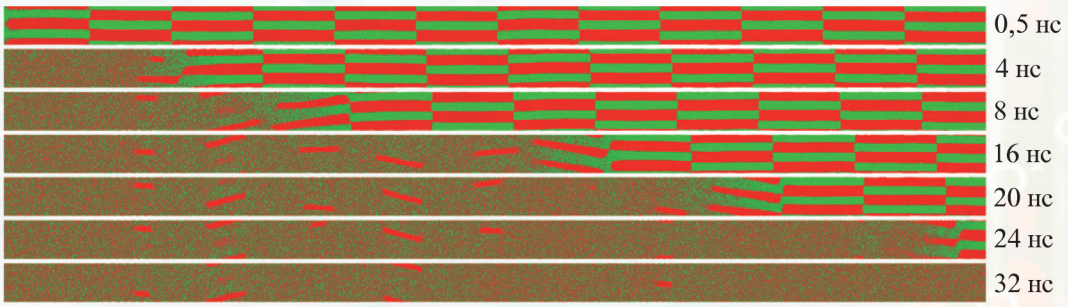


Рис. 5. Снимки с распределением атомов Ti (красные точки) и атомов Al (зеленые точки) в сечении СБС, соответствующие последовательным моментам времени продвижения волны горения

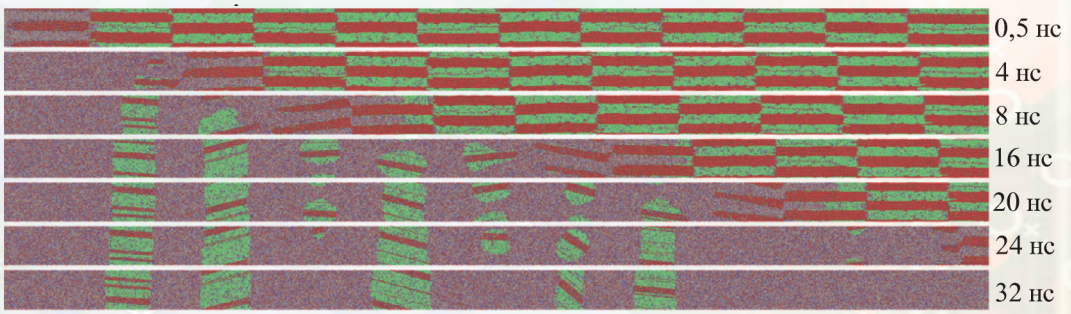


Рис. 6. Снимки с распределением различных типов элементарных ячеек в сечении СБС в последовательные моменты времени: зеленые точки – тип ГЦК-fcc; синие точки – тип ОЦК-bcc; красные точки – тип ГПУ-hcp; желтые точки – тип ico; белые точки – другие типы

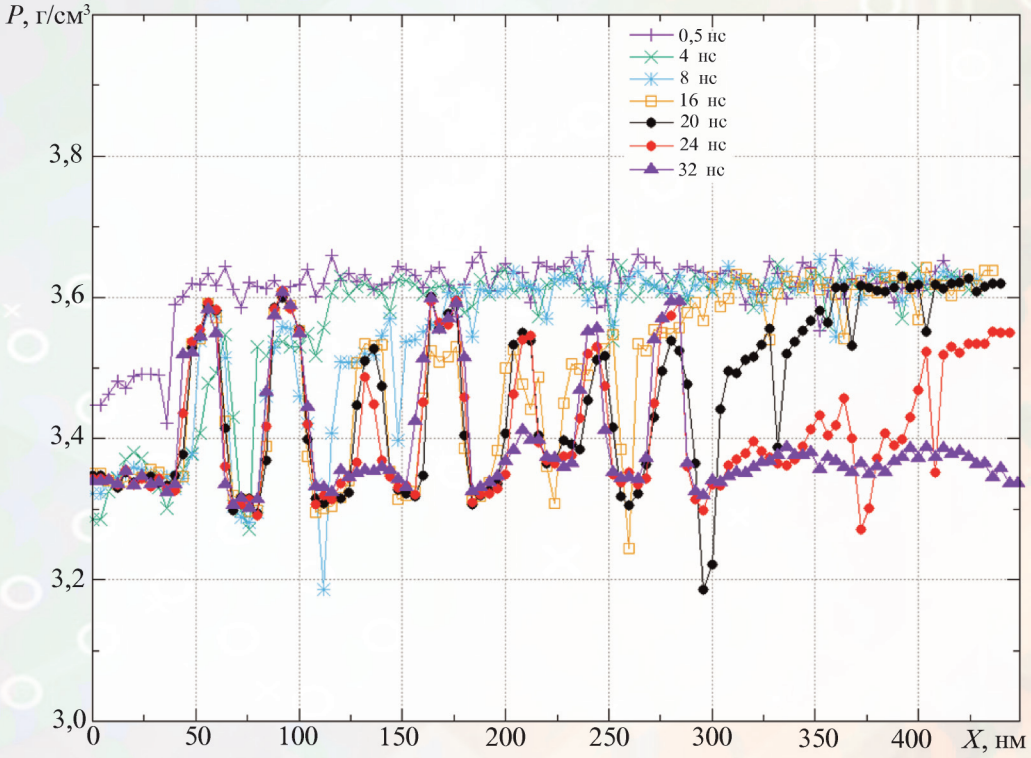
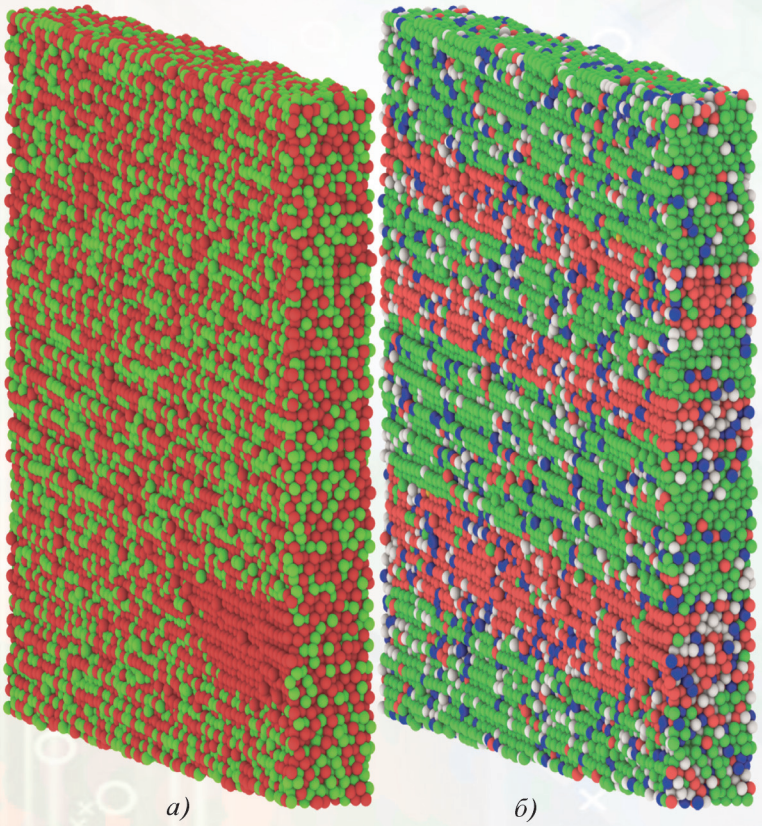


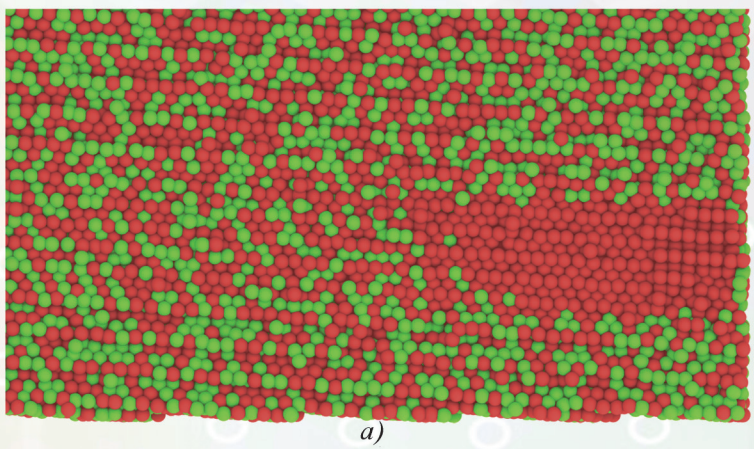
Рис. 7. Профили плотности вещества, соответствующие последовательности моментов времени при продвижении волны горения в СБС системы Ti-Al



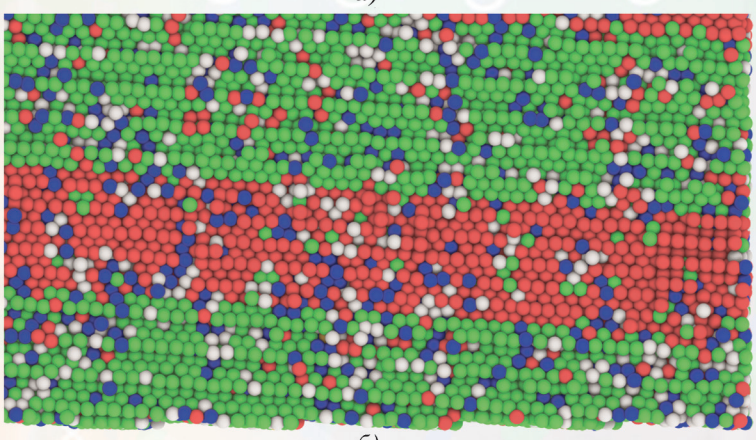
а) *б)*

Рис. 8. Соответствующие друг другу 3D-снимки гетерофазной структуры с объемными распределениями:
а – объемное распределение атомов Ti (красные шары) и атомов Al (зеленные шары);
б – объемное распределение различных типов элементарных ячеек (ГЦК-fcc – зеленые шары; ОЦК-bcc – синие шары; ГПУ-hcp – красные шары; другой тип – белые шары)

Рисунки к статье
В. И. Иордана, И. А. Шмакова
«3D-ВИЗУАЛИЗАЦИЯ ГЕТЕРОФАЗНЫХ ИНТЕРМЕТАЛЛИДНЫХ СТРУКТУР НА ОСНОВЕ КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ СВС В НАНОРАЗМЕРНОЙ СЛОИСТО-БЛОЧНОЙ СТРУКТУРЕ Ti-Al»



а)



б)

Рис. 9. Соответствующие друг другу 3D-снимки нижней половины гетерофазной структуры, отображенной на рис. 8:
а – объемное распределение атомов Ti (красные шары) и атомов Al (зеленые шары);
б – объемное распределение различных типов элементарных ячеек (ГЦК-fcc – зеленые шары; ОЦК-bcc – синие шары; ГПУ-hcp – красные шары; другой тип – белые шары)

ISSN 2220-3397. Программная инженерия. 2022. Том 13. № 10. 469–524. Подписной индекс по Объединенному каталогу "Пресса России" – 22765