УДК 519.685+519.682.2+004.032.22

Принципы построения системы детерминированного параллельного программирования

Адамович Алексей Игоревич
Старший научный сотрудник,
Институт программных систем им. А.К. Айламазяна РАН,
152021, Ярославская обл., Переславский район, с. Веськово, ул. Петра Первого, д. 4 «а»,
e-mail: lexa@adam.botik.ru
Климов Андрей Валентинович
Старший научный сотрудник,
Институт прикладной математики им. М.В. Келдыша РАН,
125047, г. Москва, Миусская пл., д. 4, e-mail: klimov@keldysh.ru

Аннотация. В связи с взрывным ростом сложности программ для многоядерных процессоров и суперкомпьютеров в последнее десятилетие приобретает популярность и становится всё более актуальной идея параллельных вычислений с детерминированностью, гарантированной языком и системой программирования. В докладе анализируется проблема, как сделать параллельное программирование как можно более детерминированным. Дается обзор некоторых подходов к ее решению. Описываются принципы построения системы программирования, разрабатываемой авторами, предоставляющей возможность писать как детерминированный, так и недетерминированный код с гарантиями прикладному программисту, что его программа будет детерминированной. Характеризуются некоторые показательные классы задач, реализуемые в данной системе.
Ключевые слова: модели параллельных вычислений, детерминированные программы, функциональное программирование, объектно-ориентированное программирование, монотонные объекты

Введение

Параллельные и конкурентные (*англ*. concurrent) программы в общем случае недетерминированные — дают разные результаты при нескольких прогонах, так как для эффективной реализации на современной аппаратуре требуется явное использование таких средств программирования, как процессы, потоки, треды (*англ*. threads), читающие и изменяющие общие ресурсы и дающие разные результаты при различном порядке доступа потоков к ресурсам. Отладка, модификация, сопровождение таких программ намного более трудозатратны, чем привычных массовым программистам детерминированных последовательных программ. Поэтому многие языки высокого уровня «прячут» от «обычного» программиста средства конкурентного программирования на уровень реализации и в библиотеки, разрабатываемые экспертами. Однако, такие решения ограничивают изобразимые на этих языках классы программ и вынуждают писать менее эффективные программы, не масштабируемые при увеличения числа процессоров, ядер и других аппаратных средств параллельного исполнения.

Таким образом, мы имеем следующую ситуацию. Во-первых, детерминированные языки параллельного и конкурентного программирования существуют и развиваются (в качестве яркого примера приведем чисто функциональных язык Haskell и его библиотеки для параллельного программирования). Во-вторых, нет и не может быть одного языка и библиотеки детерминированного параллельного программирования, который удовлетворил бы все потребности, даже если зафиксировать круг языковых понятий конкурентного программирования, который был бы базой для детерминированного, например, объектно-ориентированные

Наукоемкое программное обеспечение : труды семинара 12-й Междунар. Ершовской конф. по информатике (ПСИ'19). 2–3 июля 2019 г. /
Ин-т систем информатики им. А. П. Ершова СО РАН, Новосиб. гос. ун-т. — Новосибирск : ИПЦ НГУ, 2019. — С. 26–33. — ISBN 978-5-4437-0909-3.

языки типа Java с понятием тредов (*англ*. threads). В результате значительно разнесены уровень детерминированного программирования, считающийся «высоким», и уровень реализации языков и библиотек, считающийся «низким». На этих уровнях используются сильно различающиеся языки и инструменты и требуется очень разная квалификация разработчиков — столь же отличающиеся, как, например, у разработчиков компиляторов и систем программирования и их пользователей.

Возникает вопрос: на сколько можно приблизить эти уровни? Нельзя ли в рамках одного языка программирования (пусть это будет объектно-ориентированный язык типа Java) дать и универсальные средства недетерминированного программирования для создания базовых инструментов и библиотек, и определить «высокий» уровень, подъязык, проверяемый компилятором, при программировании на котором гарантируется детерминированность. «Нижний» универсальный уровень потребуется для постоянного расширения и развития проблемно-ориентированных библиотек для реализации определенных классов алгоритмов. Здесь требуются повышенные трудозатраты для отработки библиотек. Но потом они используются не в одной, а во многих прикладных программах данного класса, программирование и отладка которых намного легче и дешевле благодаря детерминированности, гарантированной авторами библиотек и системы программирования.

Авторы данного доклада ставят цель дать положительный ответ на этот вопрос в виде двухуровневой системы программирования на языке типа Java (реализованному на виртуальной Java-машине, JVM). Нижний, базовый уровень — это язык Java, на котором определяются классы, используемые на «верхнем» уровне. Верхний уровень — это Java-подобный язык (или Java-подмножество), напоминающий чисто функциональным язык с естественной (для функциональных языков) параллельной реализацией и с объектно-ориентированными расширениями, обеспечивающими использование классов «нижнего» уровня, не нарушая детерминированности параллельных вычислений. Библиотечные классы нижнего уровня и их объекты мы называем *монотонными* (по причинам, которые объясним ниже). Проект продолжает наши работы по Т-системе и монотонным объектам [1–6].

Данный доклад основан на статье авторов [7], содержащей обзор основных средств детерминированного параллельного программирования, разработанных к настоящему времени, и нашими предложениями по дальнейшему развитию этих средств. Здесь мы даем «выжимку» этого обзора, приводим обоснование и архитектуру разрабатываемой нами системы и кратко характеризуем два класса задач, на которых продемонстрируем методику создания детерминированных параллельных программ в наших будущих работах.

1. Обзор средств детерминированного параллельного программирования

1.1 Детерминированный параллелизм чисто функциональных языков

Отправной точкой, общей основой многих языков со средствами параллельного исполнения являются функциональные языки, которые в своем «чистом» виде не имеют побочных эффектов и потому «легко» распараллеливаются. Распараллеливание кода на функциональном языке, конечно, подразумевает сохранение семантики языка, то есть эквивалентность результата параллельного исполнения каждой программы эталонному последовательному, а также денотационной семантике, тем самым — детерминированность.

Максимально параллельное исполнение функциональной программы делается вызовами каждой функции в своем параллельном процессе. Это отнюдь не является эффективным решением в общем случае: возникает проблема, наоборот, ограничить параллелизм, чтобы эффективно использовать ограниченные аппаратные ресурсы, — то есть не «распараллелить», а «секвенциализировать» (*англ*. sequentialize), объединить потенциально параллельные группы вычислений в последовательный поток управления, тред. Прагматичное решение

(которому мы следуем в нашем проекте) — дать программисту языковые средства обозначать, где параллельные вызовы, а где последовательный код. Если двигаться с другой стороны — от последовательных языков к параллельным, то аналогичное решение наблюдаем, когда вводится конструкция (или библиотечные средства), называемая в ряде языков «future»: вызов функции в отдельном процессе, исполняемом параллельно до тех пор, пока вызвавшему процессу не понадобится результат и он не встанет на ожидание завершения.

Большинство языков, называемых функциональными, — «грязные», то есть позволяют побочные эффекты и не прячут понятие параллельного процесса, треда, позволяя явно управлять ими, так же как и в массовых объектно-ориентированных языках со средствами параллелизма. Из распространенных функциональных языков лишь Haskell сохраняет «чистоту» и осторожно расширяется средствами параллелизма — как правило, библиотечными с соответствующей поддержкой на нижнем уровне в реализации языка [8].

Среди большого разнообразия публикаций по этому направлению особенно интересен сборник, подводящий итоги по состоянию на конец 1990-х годов [9]. Не выходя за рамки функциональной модели вычислений, также строятся проблемно-ориентированные декларативные языки для конкретных областей применения. Например, таким является язык Норма [10] для решения сеточных задач некоторого класса из математической физики.

С другой стороны, современные объектно-ориентированные языки содержат в качестве своего подмножества чисто функциональный язык, на котором можно программировать в функциональном стиле без побочного эффекта. Таковы Java, C#, Kotlin, JavaScript и другие. Здесь основное неудобство — то, что компилятор не проверяет принадлежность функциональному подмножеству и соблюдение этого условия — дело программиста.

1.2 Неизменяемые объекты, immutability

Если двигаться со стороны императивных и объектно-ориентированных языков, существуют ряд работ по введению ограничений, проверяемых компилятором и/или во время исполнения и обеспечивающих нужные свойства. Если потребовать, чтобы все объекты были неизменяемыми (*англ*. immutable), то есть их состояние не менялось после отработки инициализаторов, то объектно-ориентированный язык практически превращается в функциональный, которому присущ детерминированный параллелизм.

На практике трудно обходиться совсем без изменений и побочных эффектов, поэтому понятию неизменяемости (*англ*. immutability) придают различную степень «строгости». Статья [11] содержит хороший обзор работ по этой теме.

1.3 Статические методы обеспечения детерминированности

Имеется много работ по статическому анализу кода, содержащему побочный эффект, имеющих целью выявить частные случаи, когда параллельное исполнение сохраняет детерминированность результата из-за отсутствия «гонок». Из ряда работ, упомянутых в нашем обзоре [7], отметим разработку языка Deterministic Parallel Java, DPJ [12] как имеющую прагматическую цель в рамках популярного объектно-ориентированного языка. В наших работах эти результаты не используются, и мы подходим к задаче с другой стороны.

1.4. Обеспечение детерминированности операциями над данными

Следующий подход к обеспечению детерминированности параллельных программ, в отличие от методов предыдущего раздела, не использует никаких статических средств анализа. За основу берется чисто функциональный язык программирования, быть может, со всевозможными расширениями, не нарушающими функциональность и распараллеливаемость. Затем для взаимодействия параллельных процессов предоставляются специальные структу-

ры данных с так определенными операциями, чтобы не нарушалась детерминированность. Оказывается, это не только возможно, но такая идея породила целое направление, например:

- I-структуры (*англ*. I-structures) [13];
- сети Кана (*англ*. Kahn networks) [14];
- TStreams, Concurrent Collections [15];
- структуры данных на решетках (*англ*. lattice-based data structures) [16,17].

У этих подходов есть общая черта: переменные, объекты, через которые осуществляется взаимодействие параллельных процессов, меняют свое состояние монотонно, только вверх на некоторой полурешетке от неопределенного состояния ($\bot$) к «всё более определенному». При этом верхний элемент решетки ($\top$) обозначает «переопределено»; в программе это соответствует ошибке, выработке исключения. Например, множество значений I-структур с целыми числами описывается решеткой, называемой «плоской», состоящей из нижнего элемента «не определено» ($\bot$), не сравнимых между собой целых чисел и верхнего элемента «переопределено» ($\top$). При выполнении операции присваивания значения y в переменную со значением x в нее записывается наименьшая верхняя грань значений x и y. Если полученный результат оказывается верхним элементом $\top$, то вырабатывается исключение.

Эта идея в общем виде была проработана в диссертации Lindsey Kuper [16] и в публикациях вместе с ее коллегами [17]. Она доказала детерминированность параллельных вычислений для процессов, взаимодействующих через переменные, принимающие значения из произвольной (полу)решетки.

В нашем проекте мы используем эти идеи, обобщая их на объекты, определяемые пользователем, с монотонно изменяющимся состоянием.

2. Архитектура двухуровневой системы детерминированного параллельного программирования

Поскольку не существует единого набора средств, зафиксированных в языке программирования и библиотеке, обеспечивающих все случаи детерминированного параллельного программирования, входной язык должен позволять как детерминированный, так и недетерминированный код. Но чтобы прикладной программист мог ограничивать себя средствами гарантированно детерминированного программирования, входной язык системы должен быть двухуровневым, то есть код должен четко подразделяться на две части — гарантированно детерминированную и потенциально недетерминированную:

- верхний уровень — детерминированная часть: прикладной код на подмножестве языка, который пишет эксперт в данной предметной области. Ему гарантируется детерминированность любой его программы, использующей заготовленные библиотеки нижнего уровня. Принадлежность детерминированному подмножеству проверяется компилятором. Оно примерно соответствует функциональному подмножеству языка Java и ему подобных;
- нижний уровень — недетерминированная часть: библиотеки классов, создаваемые квалифицированными программистами для определенных областей применения на универсальном объектно-ориентированном языке (типа Java) с богатым набором изобразительных средств, позволяющем кодировать недетерминированные параллельные алгоритмы. Авторы библиотек гарантируют детерминированность параллельных программ их пользователям, кодирующим на функциональном подмножестве языка на верхнем уровне. Такие классы и объекты мы называем *монотонными*, поскольку, как и в работах упомянутых в предыдущем разделе 1.4, их состояние меняется монотонно на некоторой (полу)решетке, которую, в принципе, можно построить по коду класса. В будущем мы рассчитываем, что удастся разработать средства автоматизации доказательств в подавляющем числе случаев, что опреде-

ление данного класса или группы класса является монотонным и гарантирует детерминированность.

Входным языком, в принципе, может быть любой объектно-ориентированный язык со средствами параллельного и конкурентного программирования, у которого можно выделить функциональное подмножество. Такими являются большинство современных объектно-ориентированных языков. Однако в таком случае некоторые детали эффективной реализации будут «торчать» и загромождать прикладной код. Поэтому мы разрабатываем специализированный Java-подобный язык, названный Ajl [3], в котором эти «детали» будут прикрыты синтаксическим сахаром и генерироваться из привычного кода. Таким образом, мы получаем степень свободы предложить пользователям разные механизмы реализации параллельных процессов, тредов, легких тредов (*англ*. light-weight thread), файберов (*англ*. fiber), например, использовать (или не использовать) стренды (*англ*. strand) библиотеки системы Quasar.[1]

3. Примеры классов задач

Для изучения и демонстрации возможностей программирования с монотонными объектами в настоящее время мы обкатываем систему на следующих двух классах задач:

- Порождение и обработка графов. Ценность этого класса задач в том, что классические чисто функциональные языки не дают возможности обрабатывать графы эффективно, когда узлы (и, быть может, дуги) представлены объектами, а связи между ними — ссылками в полях объектов. В функциональных языках можно эффективно представлять только деревья. В системе программирования с монотонными объектами мы сохраняем большинство положительных свойств функциональных языков (детерминированность — лишь одно из них) и расширяем области эффективно представимых данных.
- Переборные алгоритмы на графах типа поиска кратчайшего пути методом ветвей и границ (*англ*. branch-and-bound). Они интересны тем, что по смыслу задачи в них присутствуют несколько видов монотонно изменяемых данных: монотонно растут пути, причем их много и нужно их отличать друг от друга; монотонно изменяется рекорд (кратчайший путь, найденным к настоящему времени), причем порядок его изменения зависит от порядка исполнения параллельных процессов, а результат не зависит. Кроме того, необходима эквивалентность реализаций с полным перебором и с «отсевом» (*англ*. pruning) вариантов, которые заведомо не улучшат рекорд. Гарантии монотонности и эквивалентности должны быть реализованы на «нижнем уровне» в как можно меньшем по размеру коде специально сконструированных монотонных классов. Здесь появляются «объекты высшего порядка», методы которых принимают лямбда-выражения, описывающие продолжения вычислений, которые могут подвергнуться «отсеву».

Более подробная информация о методике программирования этих классов задач будет опубликована отдельно в наших будущих работах.

Заключение

В докладе дана мотивировка и принципы построения системы детерминированного параллельного программирования, разрабатываемой авторами. Представлен краткий обзор предшествующих работ, идеи которых используются в данной разработке. Охарактеризованы два класса прикладных задач, на которых система отрабатывается и будет продемонстрирована в наших будущих публикациях.

[1] https://github.com/puniverse/quasar

СПИСОК ЛИТЕРАТУРЫ

1. С.М. Абрамов, А.И. Адамович, М.Р. Коваленко. Т-система — среда программирования с поддержкой автоматического динамического распараллеливания программ. Пример реализации алгоритма построения изображений методом трассировки лучей // Программирование, 25:2, 1999. С. 100–107.
2. А.И. Адамович. Струи как основа реализации понятия Т-процесса для платформы JVM // Программные системы: теория и приложения, 6:4 (27), 2015. С. 177–195. DOI: 10.25209/2079-3316-2017-8-4-221-244.
3. А.И. Адамович. Язык программирования Ajl: автоматическое динамическое распараллеливание для платформы JVM // Программные системы: теория и приложения, 7:4 (31), 2016. С. 83–117. DOI: 10.25209/2079-3316-2016-7-4-83-117.
4. А.И. Адамович, Анд.В. Климов. Об опыте использования среды метапрограммирования Eclipse/TMF для конструирования специализированных языков // Научный сервис в сети Интернет. Труды XVIII Всероссийской научной конференции. М.: ИПМ им. М.В. Келдыша, 2016. С. 3–8. DOI: 10.20948/abrau-2016-45.
5. And.V. Klimov. Dynamic Specialization in Extended Functional Language with Monotone Objects // SIGPLAN Not., 26:9, 1991. P. 199–210. DOI: 10.1145/115865.376287.
6. Анд.В. Климов. Детерминированные параллельные вычисления с монотонными объектами // Научный сервис в сети Интернет: многоядерный компьютерный мир. Труды Всероссийской научной конференции. М.: Изд-во Московского университета, 2007. С. 212–217.
7. А.И. Адамович, Анд.В. Климов. Как создавать параллельные программы, детерминированные по построению? Постановка проблемы и обзор работ // Программные системы: теория и приложения, 2017, 8:4(35). С. 221–244. DOI: 10.25209/2079-3316-2017-8-4-221-244.
8. S. Marlow. Parallel and Concurrent Programming in Haskell. O'Reilly, CA, USA, 2013.
9. K. Hammond, G. Michelson (eds.). Research Directions in Parallel Functional Programming, Springer, London, UK, 1999.
10. А.Н. Андрианов, Т.П. Баранова, А.Б. Бугеря, К.Н. Ефимкин. Трансляция непроцедурного языка Норма для графических процессоров // Препринты ИПМ им. М.В. Келдыша, 2016, № 73. 24 с. DOI: 10.20948/prepr-2016-73.
11. A. Potanin, J. Östlund, Y. Zibin, M.D. Ernst. Immutability // Aliasing in Object-Oriented Programming, eds. D. Clarke, J. Noble, T. Wrigstad. LNCS 7850. Springer, 2013. P. 233–269.
12. R.L. Bocchino (Jr.), V.S. Adve, S.V. Adve, M. Snir. Parallel Programming Must Be Deterministic by Default // Fifth USENIX Conference on Hot Topics in Parallelism, HotPar'09. USENIX Association, 2009. P. 4–4 (6 pages).
13. Arvind, R.S. Nikhil, K.K. Pingali. I-structures: Data Structures for Parallel Computing // ACM Trans. Program. Lang. Syst., 11:4, 1989. P. 598–632. DOI: 10.1145/69558.69562.
14. G. Kahn. The Semantics of a Simple Language for Parallel Programming // IFIP Congress, 1974. P. 471–475.
15. M.G. Burke, K. Knobe, R. Newton, V. Sarkar. Concurrent Collections Programming Model // Encyclopedia of Parallel Computing, ed. D. Padua. Springer US, 2011. P. 364–371. DOI: 10.1007/978-0-387-09766-4_238.
16. L. Kuper. Lattice-based Data Structures for Deterministic Parallel and Distributed Programming, Ph.D. Thesis, 2015.
17. L. Kuper, A. Todd, S. Tobin-Hochstadt, R.R. Newton. Taming the Parallel Effect Zoo: Extensible Deterministic Parallelism with LVish // ACM SIGPLAN Not., 49:6, 2014. P. 2–14. DOI: 10.1145/2666356.2594312.

UDK 519.685+519.682.2+004.032.22

PRINCIPLES OF CONSTRUCTING A DETERMINISTIC PARALLEL PROGRAMMING SYSTEM

Alexei I. Adamovich
Senior researcher, Ailamazyan Program Systems Institute of Russian Academy of Sciences
4a, Peter I str., selo Veskovo, Pereslavl region, Yaroslavl oblast, 152021, Russia,
e-mail: lexa@adam.botik.ru
Andrei V. Klimov
Senior researcher, Keldysh Institute of Applied Mathematics of Russian Academy of Sciences
4, Miusskaya sq., 125047, Moscow, Russia
e-mail: klimov@keldysh.ru

Annotation. Due to the explosive growth of complexity of programs for multi-core processors and supercomputers, the idea of parallel computing with determinism guaranteed by the programming language and system is becoming increasingly significant. This paper analyzes the problem of how to make parallel programming as deterministic as possible. An overview of some approaches to its solution is given. It describes the principles of constructing a programming system developed by the authors, which provides the opportunity to write both deterministic and non-deterministic code with guarantees to the application programmer that his program will be deterministic. Some representative classes of tasks implemented in this system are characterized.

Keywords: models of parallel computation, deterministic programs, functional programming, object-oriented programming, monotonic objects

References

1. S.M. Abramov, A.I. Adamovich, M.R. Kovalenko. T-sistema — sreda programmirovanija s podderzhkoj avtomaticheskogo dinamicheskogo rasparallelivanija programm. Primer realizacii algoritma postroenija izobrazhenij metodom trassirovki luchej [T-System — An Environment Supporting Automatic Dynamic Parallelization of Programs: An Example of the Implementation of an Image Rendering Algorithm Based on the Ray Tracing Method]. Programmirovanie = Programming, 25:2, 1999, pp. 100–107. (In Russian).
2. A.I. Adamovich. Strui kak osnova realizacii ponjatija T-processa dlja platformy JVM [Fibers as the basis for the implementation of the notion of the T-process for the JVM platform]. Programmnye sistemy: teorija i prilozhenija = Program Systems: Theory and Applications, 6:4 (27), 2015, pp. 177–195, DOI: 10.25209/2079-3316-2017-8-4-221-244. (In Russian).
3. A.I. Adamovich. Jazyk programmirovanija Ajl: avtomaticheskoe dinamicheskoe rasparallelivanie dlja platformy JVM [The Ajl programming language: the automatic dynamic parallelization for the JVM platform]. Programmnye sistemy: teorija i prilozhenija = Program Systems: Theory and Applications, 7:4 (31), 2016, pp. 83–117. DOI: 10.25209/2079-3316-2016-7-4-83-117. (In Russian).
4. A.I. Adamovich, And.V. Klimov. Ob opyte ispol'zovanija sredy metaprogrammirovanija Eclipse/TMF dlja konstruirovanija specializirovannyh jazykov [On Experience of Using the Metaprogramming Development Environment Eclipse/TMF for Construction of Domain-Specific Languages]. Nauchnyj servis v seti Internet. Trudy XVIII Vserossijskoj nauchnoj konferencii. Moscow: IPM im. M.V. Keldysha, 2016, pp. 3–8. DOI: 10.20948/abrau-2016-45. (In Russian).
5. And.V. Klimov. Dynamic Specialization in Extended Functional Language with Monotone Objects. SIGPLAN Not., 26:9, 1991, pp. 199–210. DOI: 10.1145/115865.376287.
6. And.V. Klimov. Determinirovannye parallel'nye vychislenija s monotonnymi ob"ektami [Deterministic parallel computation with monotonic objects]. Nauchnyj servis v seti Internet: mnogojadernyj komp'juternyj mir. Trudy Vserossijskoj nauchnoj konferencii. Moscow, MSU Publ., 2007, pp. 212–217. (In Russian).

7. A.I. Adamovich, And.V. Klimov. Kak sozdavat' parallel'nye programmy, determinirovannye po postroeniju? Postanovka problemy i obzor rabot [How to create deterministic by construction parallel programs? Problem statement and survey of related works]. Programmnye sistemy: teorija i prilozhenija = Program Systems: Theory and Applications, 2017, 8:4(35), pp. 221–244. DOI: 10.25209/2079-3316-2017-8-4-221-244. (In Russian).
8. S. Marlow. Parallel and Concurrent Programming in Haskell. O'Reilly, CA, USA, 2013.
9. K. Hammond, G. Michelson (eds.). Research Directions in Parallel Functional Programming, Springer, London, UK, 1999.
10. A.N. Andrianov, T.P. Baranova, A.B. Bugerya, K.N. Yefimkin. Transljacija neprocedurnogo jazyka Norma dlja graficheskih processorov [Nonprocedural NORMA language translation for GPUs]. Preprinty IPM im. M.V. Keldysha = Keldysh Institute Preprints, 2016, no. 73. 24 p. DOI: 10.20948/prepr-2016-73. (In Russian).
11. A. Potanin, J. Östlund, Y. Zibin, M.D. Ernst. Immutability. Aliasing in Object-Oriented Programming, eds. D. Clarke, J. Noble, T. Wrigstad. LNCS 7850. Springer, 2013, pp. 233–269.
12. R.L. Bocchino (Jr.), V.S. Adve, S.V. Adve, M. Snir. Parallel Programming Must Be Deterministic by Default. Fifth USENIX Conference on Hot Topics in Parallelism, HotPar'09. USENIX Association, 2009, pp. 4–4 (6 pages).
13. Arvind, R.S. Nikhil, K.K. Pingali. I-structures: Data Structures for Parallel Computing. ACM Trans. Program. Lang. Syst., 11:4, 1989, pp. 598–632. DOI: 10.1145/69558.69562.
14. G. Kahn. The Semantics of a Simple Language for Parallel Programming. IFIP Congress, 1974, pp. 471–475.
15. M.G. Burke, K. Knobe, R. Newton, V. Sarkar. Concurrent Collections Programming Model. Encyclopedia of Parallel Computing, ed. D. Padua. Springer US, 2011, pp. 364–371. DOI: 10.1007/978-0-387-09766-4_238.
16. L. Kuper. Lattice-based Data Structures for Deterministic Parallel and Distributed Programming, Ph.D. Thesis, 2015.
17. L. Kuper, A. Todd, S. Tobin-Hochstadt, R.R. Newton. Taming the Parallel Effect Zoo: Extensible Deterministic Parallelism with LVish. ACM SIGPLAN Not., 49:6, 2014, pp. 2–14. DOI: 10.1145/2666356.2594312.