AUTOMATIC PROGRAM CONSTRUCTION WITH ACCELERATORS USAGE BASED ON THE ACTIVE KNOWLEDGE CONCEPT IN LUNA SYSTEM

V. E. Malyshkin*,**,***, V. A. Perepelkin*,**,***, V. A. Spirin*,**

*Novosibirsk State University,
630090, Novosibirsk, Russia
**Institute of computational mathematics and mathematical geophysics SB RAS,
630090, Novosibirsk, Russia
***Novosibirsk State Technical University,
630073, Novosibirsk, Russia

DOI: 10.24412/2073-0667-2025-4-73-88
EDN: ZCPPTC

As accelerators are designed for specific types of computations (e. g., GPUs, NPUs, FPGAs), they can significantly improve the performance of corresponding programs. In fact, most of the computing power in modern multicomputers is provided by accelerators. However, using accelerators in programs requires knowledge of system programming, which makes development more difficult for experts in other domains. For example, it is necessary to manage data transfers between CPUs and accelerators, implement load balancing, and schedule computations on accelerators. Achieving good efficiency requires different optimization methods in different subject domains.

One way to reduce the labor costs of developing programs that use accelerators is to offload some complex routine tasks from humans to an automation system. Such systems can lower the required level of competence for developing efficient programs by performing automatic program construction. Potentially, these automation systems can also produce programs with higher efficiency than average hand-coded implementations. Since no universal automation approach exists for such systems, it is necessary to develop various approaches for particular classes of applied problems and types of accelerators. One method that can be applied here is the active knowledge concept [1].

Automatic construction of parallel programs using the active knowledge concept is performed based on computational models (CMs). CMs are designed to represent knowledge in a certain subject domain. For simplicity, a CM can be viewed as a bipartite directed graph consisting of operations and variables. Each variable represents a certain value within the subject domain. Each operation represents the derivation of its output variables (on outgoing arcs) from its input variables (on incoming arcs). Operations are supplied with code fragments (CFs) in the form of conventional subroutines (procedures). To solve a problem in this CM's subject domain, we define a VW-task on the CM, where V is the set of input variables and W is the set of output variables to be computed. A VW-plan is then derived from the VW-task (if it exists). A VW-plan is a subgraph of the CM containing the necessary operations and variables to compute the variables in W from the variables in V. Using the VW-plan, a program can be constructed that takes values for the variables in V and computes values for the intermediate variables and the variables in W. Such programs can be parallel if different variables can be evaluated independently.

This research was carried out under the state contract with ICM&MG SB RAS FWNM-2025-0005.

The actual program construction (VW-plan derivation and/or program generation) is performed by an active knowledge system. An example of such a system, explored in this paper, is the LuNA system [2, 3]. The LuNA system takes a prepared VW-plan and performs program generation; the generated program is then executed by the runtime system. The runtime system uses a thread pool to schedule and execute operations. The thread pool consists of one queue and multiple worker threads that take available tasks from the queue and execute them. In a distributed system, multiple instances of the runtime system can execute the generated program. These instances communicate to decide which operation will be executed on which node.

In this paper, we consider an extension of the LuNA runtime system to support accelerators. In particular, we examine the Huawei Ascend neural processor [4]. We propose a high-level specification called CoFaNA (Code Fragment Notation for Accelerators) for describing computations to be executed on the Ascend accelerator. We also propose an extension of the thread pool to perform computations executed on both CPUs and Ascends. The thread pool was extended to have three queues: one for tasks to be executed on CPUs, another for tasks to be executed on Ascends, and a third for heterogeneous tasks that can be executed on either CPUs or Ascends. We also split the worker threads into two groups: one group can execute tasks on CPUs (including heterogeneous tasks), and the other group can execute tasks on Ascends.

Next, based on the implementation of support for the Ascend accelerator, we propose a subsystem that can potentially use any accelerator. By implementing support for a particular accelerator in a separate extension module (plugin), the subsystem can use subprograms from this module during program generation and execution in the runtime system. An extension module implements interfaces for the code generator, context, and thread pool, and also provides metainformation. This metainformation is used during program generation to link necessary shared libraries. The context is used to initialize or deinitialize libraries used in the extension module. The code generator can parse various high-level specifications (for example, the CoFaNA format for the Ascend accelerator). The thread pool can be implemented in various ways to use accelerators more efficiently.

An experimental study of using the Ascend accelerator was carried out on an Atlas 800 server [11]. We considered the problems of block multiplication of dense matrices and convolution of seismic traces [10]. The results show that LuNA implementations have execution times comparable to hand-coded C++ programs. The CoFaNA format helped to reduce the labor costs required to develop programs that use the Ascend accelerator.

In conclusion, this research explored the automatic construction of programs that use accelerators, based on the active knowledge concept. As a result of the work, the LuNA system was extended to support the Huawei Ascend accelerator and potentially other accelerators.

In the future, we plan to implement support for other accelerators and, if necessary, expand the subsystem. We also plan to research improvements for using the Huawei Ascend processor in the LuNA system: expanding the CoFaNA format, considering alternative implementations of the thread pool, and automatically selecting the precision of floating-point numbers based on the properties of the applied problem being solved.

Key words: active knowledge, LuNA system, accelerator, Huawei Ascend processor, automatic program construction, high-level specification, subsystem for accelerators support.

References

1. Malyshkin V. E. Active Knowledge, LuNA and Literacy for Oncoming Centuries // LNCS, V. 9465. 2015. P. 292–303. DOI: 10.1007/978-3-319-25527-9_19.

2. Malyshkin V. E., Perepelkin V. A. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem // Proceedings of the 11th International Conference on Parallel

Computing Technologies (PaCT-2011), LNCS, 2011. V. 6873, P. 53–61. DOI: 10.1007/978-3-642-23178-0_5.

3. Malyshkin V. E., Perepelkin V. A. Postroenie baz aktivnyh znanij dlya avtomaticheskogo konstruirovaniya reshenij prikladnyh zadach na osnove sistemy LuNA // Parallel'nye vychislitel'nye tekhnologii — XVIII vserossijskaya nauchnaya konferenciya s mezhdunarodnym uchastiem, PaVT'2024, g. Chelyabinsk, 2–4 aprelya 2024 g. Korotkie stat'i i opisaniya plakatov. Chelyabinsk: Izdatel'skij centr YUUrGU, 2024. P. 57–68. DOI: 10.14529/pct2024 (In Russian).

4. Liang X. Ascend AI Processor Architecture and Programming: Principles and Applications of CANN. Elsevier, 2020. DOI: 10.1016/C2020-0-00270-7.

5. Malyshkin V., Perepelkin V., Schukin G. Scalable Distributed Data Allocation in LuNA Fragmented Programming System // The Journal of Supercomputing, 2017. V. 73, Iss. 2, Springer. P. 726–732. DOI: 10.1007/s11227-016-1781-0.

6. Malyshkin V. E., Perepelkin V. A., Schukin G. A. Distributed Algorithm for Data Allocation in Luna Fragmented Programming System // "Problems of informatics". 2017, N 1. P. 74–88.

7. Using Operator Samples [Electron. Res.]: `https://www.hiascend.com/document/detail/zh/CANNCommunityEdition/82RC1alpha001/opdevg/tbeaicpudevg/atlasopdev_10_0025.html`.

8. AscendCL architecture and basic concepts [Electron. Res.]: `https://www.hiascend.com/document/detail/zh/CANNCommunityEdition/82RC1alpha001/appdevg/aclcppdevg/aclcppdevg_000004.html`.

9. Spirin V. A. Razrabotka algoritmov avtomatizacii primeneniya NPU v sisteme LuNA // Parallel'nye vychislitel'nye tekhnologii — XVIII vserossijskaya nauchnaya konferenciya s mezhdunarodnym uchastiem, PaVT'2024, g. Chelyabinsk, 2–4 aprelya 2024 g. Korotkie stat'i i opisaniya plakatov. Chelyabinsk: Izdatel'skij centr YUUrGU, 2024. P. 165–176. DOI: 10.14529/pct2024 (In Russian).

10. Khairetdinov M. S. Algoritmy potochnoj svertki v zadachah aktivnogo vibrosejsmoakusticheskogo monitoringa / M. S. Khairetdinov, G. M. Voskobojnikova, G. S. Seduhina // Geosibir', 2017 (In Russian).

11. Laboratoriya iskusstvennogo intellekta i informacionnyh tehnologiy (In Russian) [Electron. Res.]: `https://icmmg.nsc.ru/ru/content/pages/laboratoriya-iskusstvennogo-intellekta-i-informacionnyh-tehnologiy`.

12. Chen L. Deep learning and practice with mindspore. Springer Nature, 2021. DOI: 10.1007/978-981-16-2233-5.

13. Feng W., Maghareh R., Wang K. T. A. Extending DPC++ with Support for Huawei Ascend AI Chipset // International Workshop on OpenCL. 2021. P. 1–4. DOI: 10.1145/3456669.3456684.

14. Gu R., Becchi M. A comparative study of parallel programming frameworks for distributed GPU applications // Proceedings of the 16th ACM International Conference on Computing Frontiers. 2019. P. 268–273. DOI: 10.1145/3310273.3323071.

15. Robson M. P., Buch R., Kale L. V. Runtime coordinated heterogeneous tasks in Charm++ // 2016 Second International Workshop on Extreme Scale Programming Models and Middlewar (ESPM2). IEEE, 2016. P. 40–43. DOI: 10.1109/ESPM2.2016.011.

16. Bauer M. et al. Legion: Expressing locality and independence with logical regions // SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. IEEE, 2012. P. 1–11. DOI: 10.1109/SC.2012.71.

17. Augonnet C. et al. StarPU: a unified platform for task scheduling on heterogeneous multicore architectures // Euro-Par 2009 Parallel Processing: 15th International Euro-Par Conference, Delft, The Netherlands, August 25–28, 2009. Proceedings 15. Springer Berlin Heidelberg, 2009. P. 863–874. DOI: 10.1007/978-3-642-03869-3_80.

18. Ayguadé E. et al. An extension of the StarSs programming model for platforms with multiple GPUs // Euro-Par 2009 Parallel Processing: 15th International Euro-Par Conference, Delft, The Netherlands, August 25–28, 2009. Proceedings 15. Springer Berlin Heidelberg, 2009. P. 851–862. DOI: 10.1007/978-3-642-03869-3_79.

ОБЕСПЕЧЕНИЕ АВТОМАТИЧЕСКОГО КОНСТРУИРОВАНИЯ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ СПЕЦВЫЧИСЛИТЕЛЕЙ НА ОСНОВЕ КОНЦЕПЦИИ АКТИВНЫХ ЗНАНИЙ В СИСТЕМЕ LUNA

В. Э. Малышкин*,**,***, В. А. Перепелкин*,**,***, В. А. Спирин*,**

*Новосибирский национальный исследовательский государственный университет, 630090, Новосибирск, Россия
**Институт вычислительной математики и математической геофизики СО РАН, 630090, Новосибирск, Россия
***Новосибирский государственный технический университет, 630073, Новосибирск, Россия

УДК 004.4'242
DOI: 10.24412/2073-0667-2025-4-73-88
EDN: ZCPPTC

Спецвычислители, такие как GPU или NPU, ориентированы на определенный характер вычислительной нагрузки, что позволяет в значительной степени повысить их производительность. Разработка эффективных программ, использующих спецвычислители, является трудоемкой задачей, так как требует знаний и навыков в области системного программирования. В частности, разработка таких программ затруднительна для специалистов других профилей. В работе рассматривается автоматическое конструирование параллельных программ с использованием спецвычислителей. Автоматизация направлена на снижение трудоемкости разработки и исполнения программ с получением приемлемой эффективности. Предлагаемое в работе решение основывается на концепции активных знаний и реализуется в системе активных знаний LuNA. Рассматривается расширение архитектуры исполнительной системы для поддержки спецвычислителей на примере представителя нейронных процессоров Huawei Ascend. Поддержка спецвычислителя обеспечивается отдельным слабосвязанным модулем. Приводятся результаты экспериментального исследования решения задач блочного умножения плотных матриц и корреляционной свертки сейсмотрасс с использованием спецвычислителя.

Ключевые слова: концепция активных знаний, система LuNA, спецвычислитель, процессор Huawei Ascend, автоматическое конструирование программ, высокоуровневая спецификация, подсистема поддержки спецвычислителей.

Введение. Важную роль в современных высокопроизводительных вычислениях играют спецвычислители, такие как графические ускорители (GPU), нейронные процессоры (NPU), ПЛИС (FPGA). Спецвычислители позволяют существенно повысить производительность вычислений за счет ориентации на конкретный характер вычислительной нагрузки. В составе суперкомпьютеров именно спецвычислители, как правило, поставляют бо́льшую часть вычислительной мощности.

Работа выполнена в рамках государственного задания ИВМиМГ СО РАН FWNM-2025-0005.

Разработка эффективных параллельных программ, использующих спецвычислители, часто сопряжена с дополнительными проблемами по сравнению с использованием обычных процессоров (CPU). Тут и далее эффективность понимается с точки зрения времени выполнения программы, расхода памяти, нагрузки на сеть и т. п. характеристик. Эти проблемы проистекают из того, что эффективное использование спецвычислителя требует знания многих особенностей его работы и соответствующих технических средств, а также умения учитывать эти особенности при решении конкретной задачи. Существенно снизить сложность и трудоемкость разработки программ возможно за счет автоматизации этого процесса, когда часть работы перекладывается с человека на систему автоматизации. Также это позволяет, по крайней мере в перспективе, улучшить качество конструируемых программ по сравнению со средней ручной разработкой.

При всей сложности задачи автоматического конструирования эффективных параллельных программ, существуют подходы к ее решению, по крайней мере в частных случаях. Один из таких подходов — концепция активных знаний [1], определяющая методологию автоматического конструирования программ в конкретных предметных областях. В ее основе лежит идея построения базы активных знаний — машинно-ориентированного представления системы знаний о предметной области, обеспечивающей автоматическое конструирование достаточно эффективных для практического использования параллельных программ решения задач в этой предметной области. База активных знаний включает частичную аксиоматическую теорию предметной области, наработанные при ручном программировании в этой предметной области фрагменты кода и описание особенностей их применения для решения различных задач. Сильной стороной подхода является возможность автоматически учитывать специфичные особенности предметной области для обеспечения требуемой эффективности конструируемых программ.

Концепция активных знаний представляется перспективным подходом и для автоматизации конструирования параллельных программ, использующих спецвычислители, т. к. она позволяет переложить на систему и сложные рутинные технические детали использования спецвычислителя, и многие вопросы системного параллельного программирования. К таким вопросам относятся декомпозиция данных и вычислений, динамическое распределение нагрузки, синхронизация доступа к общим данным, совместное использование ядер центрального процессора и спецвычислителей и пр. Но реализация этой идеи требует развития как подхода, так и инструментария, представленного системой активных знаний LuNA [2, 3].

Целью работы являются исследование и частичная реализация возможностей концепции активных знаний по автоматизации использования специализированных вычислителей в конструируемых параллельных программах. В частности, рассматривается задача обеспечения поддержки спецвычислителей в системе LuNA на примере поддержки спецвычислителя Huawei Ascend [4].

Остальная часть статьи структурирована следующим образом. В разделе 1 приводится описание подхода к автоматическому использованию спецвычислителей на основе концепции активных знаний. В разделе 2 приводятся необходимые определения, связанные с системой LuNA, и описываются предлагаемые решения для поддержки спецвычислителей. В разделе 3+ представлены результаты экспериментального исследования для задач блочного умножения плотных матриц и корреляционной свертки сейсмотрасс с использованием спецвычислителя. В заключении подводятся итоги работы.

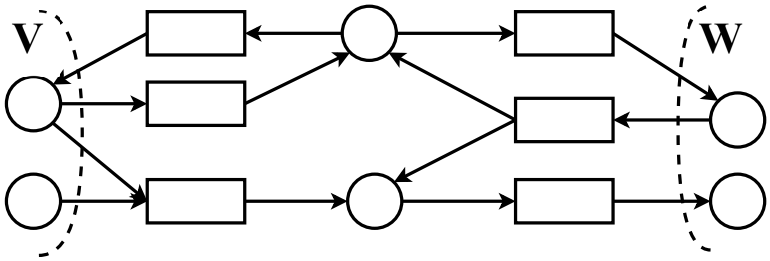


Рис. 1. Пример простой вычислительной модели, где круги представляют данные, а прямоугольники — операции; V и W представляют входные и выходные данные соответственно

1. Подход к автоматизации использования спецвычислителей. В концепции активных знаний автоматическое конструирование программ осуществляется на основе **вычислительных моделей** (ВМ). ВМ можно рассматривать как двудольный орграф, состоящий из *операций* и *переменных*, соединенных дугами (см. рис. 1.). Каждая переменная обозначает некоторую величину предметной области и может иметь значение произвольного типа. Каждая операция обозначает возможность вычислять значения своих выходных переменных (на исходящих дугах) из значений входных (на входящих дугах), и эта возможность реализуется путем предоставления **фрагмента кода** (ФК) — программного модуля (процедуры) определенного вида.

Далее, на ВМ ставится **VW-задача** — пара подмножеств переменных V и W, задающих входные и выходные переменные соответственно. По VW-задаче строится **VW-план** — подграф ВМ, который содержит достаточное количество операций и переменных, чтобы вычислить значения всех переменных из W из значений переменных из V (если такой план существует). По VW-плану строится программа, суть которой сводится к тому, чтобы считать извне значения входных переменных, вызвать в нужном порядке и с нужными аргументами ФК, вычисляя тем самым значения промежуточных и выходных переменных, после чего выдать вовне значения выходных переменных. Конструируемая программа может быть параллельной, если информационные зависимости операций это позволяют. Если конструируется программа для распределенной памяти, то система конструирования должна обеспечить передачу данных по сети тех значений переменных, которые были получены на одном узле мультикомпьютера, а потребляются — на другом (других).

Базой активных знаний называется композиция из описания ВМ и набора фрагментов кода, описания нефункциональных свойств операций, переменных и ФК, а также необходимой сопутствующей технической информации. Таким образом, база активных знаний содержит все необходимое, чтобы на ее основе полностью автоматически конструировать программы. Собственно конструирование и исполнение программ осуществляет **система активных знаний** (система LuNA [2, 3] является примером такой системы). База активных знаний служит машинно-ориентированным способом представления знаний о предметной области, полученных, в первую очередь, в результате ручного программирования специалистов в этой предметной области. Именно знания, заложенные в базу активных знаний, позволяют конструировать программы удовлетворительного качества в этой предметной области.

В качестве примера удобно рассмотреть предметную область «треугольник». Переменными обозначаются такие величины как длины сторон, углы, площадь, периметр и т. п.,

а операции соответствуют формулам вычисления одних величин из других. Если каждую формулу реализовать в виде ФК, то это позволит автоматически конструировать программы решения задач вида «дано — требуется» в этой области. Задача может решаться в несколько действий на графе ВМ, как это показано выше.

Теперь рассмотрим вопрос поддержки спецвычислителей. Наивным способом поддержать возможность конструировать программы с использованием спецвычислителей является создание таких ФК, которые внутри на основе стандартного инструментария (CUDA, OpenACC и т. п.) осуществляют вычисления на спецвычислителе. В принципе, это будет работать, но у такого подхода есть ряд недостатков. Во-первых, вся техническая работа по использованию спецвычислителя остается на пользователе, так как именно он разрабатывает ФК. Во-вторых, система «не знает» о том, что в том или ином ФК используется спецвычислитель и потому не может учитывать это при создании VW-плана и программы (для планирования ресурсов, оценки эффективности и т. д.). В-третьих, отсутствует возможность оптимизировать перемещение данных между памятью центрального процессора и памятью спецвычислителя в случае, если на спецвычислителе исполняется более одной операции.

Нормальным способом поддержать спецвычислители является добавление в систему поддержки нового вида ФК, исполняемых на спецвычислителе. Тогда система конструирования может взять на себя заботы по инициализации и освобождению ресурсов вычислителя, передаче данных (значений переменных) в память спецвычислителя и обратно, отслеживать размещение значений переменных в памяти центрального процессора и памяти спецвычислителя и оптимизировать их перемещение, планировать распределение вычислительной нагрузки по разным устройствам, выполнять синхронизацию процессов и т. п. Такая автоматизация возможна благодаря тому, что в основе конструирования лежит ВМ, явно описывающая переменные, операции и ФК.

В такой постановке задачи разрабатывать код на спецвычислителе все еще должен пользователь. Но в данном случае пользователь должен разработать только т. н. «ядро» (kernel) — только вычисления, без необходимости синхронизации процессов или передачи данных между памятями компьютера. К тому же, разработка одного модуля для спецвычислителя — это существенно проще, чем его встраивание в обычную программу.

При этом концепция активных знаний обладает преимуществами, которые распространяются и на программы со спецвычислителями. Во-первых, это возможность автоматически обеспечивать динамическую балансировку нагрузки на устройства (ядра центрального процессора и спецвычислитель). В работах [5, 6] показывалось, как на основе концепции активных знаний обеспечивается динамическая балансировка нагрузки на узлы мультикомпьютера. Этот же подход может быть использован и для спецвычислителя. Во-вторых, это дополнительные возможности отладки. Например, пользователь может обратиться к стороннему специалисту или программным инструментам для создания ФК для спецвычислителя из обычного ФК. Тогда в режиме отладки возможно автоматически сравнивать результаты работы обычного и специализированного ФК. В-третьих, это оптимизация программ на основе профилирования. Например, возможно автоматически собирать информацию о временных и прочих характеристиках исполнения ФК, после чего повторно конструировать программу с более эффективным распределением вычислений на центральных процессорах и спецвычислителях. Также в систему возможно встраивать поддержку различных продвинутых техник оптимизации вычислений (асинхронные передачи данных, техники типа CUDA Graph и т. п.) без необходимости пользователю

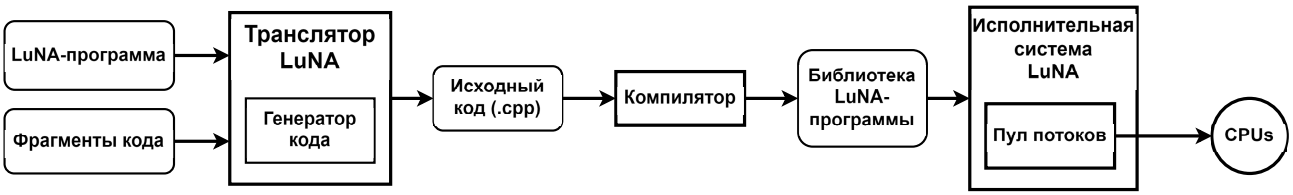


Рис. 2. Схема обработки LuNA-программы системой

даже знать о таких техниках. Эта возможность обеспечивается благодаря тому, что в базе активных знаний содержится достаточно информации для того, чтобы применять эти техники.

2. Система LuNA и поддержка спецвычислителей. Кратко рассмотрим вопрос реализации поддержки спецвычислителей в системе LuNA. Описание ВМ на языке LuNA имеет следующий вид (листинг 1).

Листинг 1. Пример LuNA-программы.

```
01:  import init_matrix(name, int);
02:  import mat_mat_mul(value, value, name, int)
                              : DEV_ID @ {annotations};
03:  import print(value, int);
04:
05:  sub main(int matSize) {
06:      df matA, matB, matC;
07:      init_matrix(matA, matSize);
08:      init_matrix(matB, matSize);
09:      mat_mat_mul(matA, matB, matC, matSize)
                                    @ {annotations};
10:      print(matC, matSize);
11:  }
```

На строках 1–3 объявляются ФК, которые определяются в других файлах в виде C++-процедур. На строках 7–10 определяются операции. Также в строке 9 определяются аннотации (после символа `@`), которые позволяют задавать нефункциональные параметры операций. Дополнительные параметры, указанные на строке 2, будут рассмотрены позже.

Файл с описанием LuNA-программы вместе с файлами с определениями ФК обрабатываются разными компонентами системы LuNA так, как это показано на рис. 2.

Поданные на вход LuNA-программа и ФК сначала обрабатываются транслятором (ключевым компонентом которого является *генератор кода*), затем компилятором языка программирования C++. В результате генерируется исполняемое представление программы в виде машинного кода, которое содержит фрагменты кода и другие процедуры, которые позволяют управлять исполнением программы.

Далее исполнительная система загружает и исполняет такое представление с использованием *пула потоков*.

В работе в качестве частного случая спецвычислителя рассматривается процессор Huawei Ascend как представитель современных нейронных процессоров. Далее в подразделе под *спецвычислителем (СВ)* подразумевается процессор Huawei Ascend.

Принцип работы с СВ. Основным понятием, описывающим вычисления для СВ, является *оператор*. Операторы описываются отдельно от прикладной программы в специальном формате [7]. На основе этого формата генерируется исполняемое представление оператора.

Система Ascend[1] поставляется со *встроенными операторами*, например, оператором GEMM (GEneral Matrix Multiplication) для умножения плотных матриц. Операторы, которые пользователи описывают сами в специальном формате, далее будем называть *собственными*.

Для взаимодействия с СВ в прикладных программах используется библиотека AscendCL [8], которая позволяет управлять памятью СВ, запускать операторы на СВ и пр.

Есть дополнительные нюансы, связанные с использованием СВ в прикладных программах. В частности, для работы со СВ требуется, чтобы поток исполнения был «привязан» к нужному СВ. Это позволяет связать вызовы процедур библиотеки AscendCL в потоке с нужным СВ (т. е. вызовы процедур по управлению памятью, запуску оператора, и пр. будут выполнены на связанном СВ).

За счет автоматизации с пользователя снимаются требования, связанные с знанием такой специфики использования СВ.

Поддержка СВ в системе LuNA. Для поддержки СВ требуется расширить возможности системы LuNA так, чтобы система могла работать со СВ без дополнительного вмешательства пользователя (инициализировать и освобождать ресурсы СВ, осуществлять передачу данных и балансировку вычислительной нагрузки между ядрами центрального процессора и СВ). Для этого был введен новый тип ФК, который описывает вычисления для СВ с соответствующим расширением синтаксиса (см. строку 2 листинга 1). Для автоматизации использования СВ был разработан алгоритм управления очередями задач и потоками в пуле (см. рис. 3). Архитектура системы была расширена (рис. 4), что позволило добавлять поддержку новых видов спецвычислителей путем добавления нового слабосвязанного модуля.

Автоматическая генерация вычислений для СВ. Одна из основных сложностей работы со СВ происходит из процесса разработки ФК для СВ. ФК должен соответствовать требованиям, которые установлены производителями СВ. При этом если требуется использовать другой спецвычислитель, то требования к формату ФК для него будут другими.

В работе предлагается формат описания вычислений CoFaNA (Code Fragment Notation for Accelerator), который позволяет описывать вычисления для СВ на высоком уровне абстракции. На основе этого формата генерируется ФК для СВ в нужном формате. Также формат CoFaNA подходит для генерации ФК для других видов спецвычислителей.

Структура формата CoFaNA представлена на листинге 2 ниже.

Листинг 2. Структура формата CoFaNA.

```
01: name OperatorName
02: types t_floats float16 float double
03: types ts_ints int
04: localvar input t_floats X0local [-1] size=sX0local
```

[1]Под системой Ascend подразумевается некоторое программное обеспечение, которое позволяет регулировать работу процессоров Ascend.

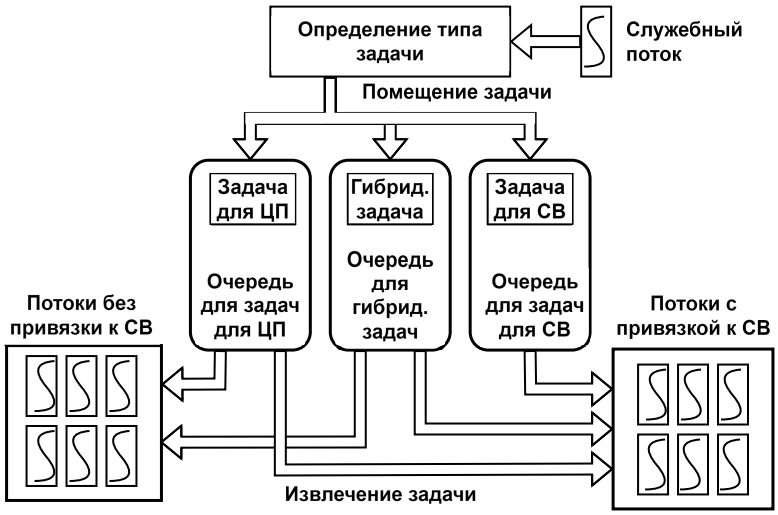


Рис. 3. Схема распределения задач с использованием трех очередей для каждого типа задач

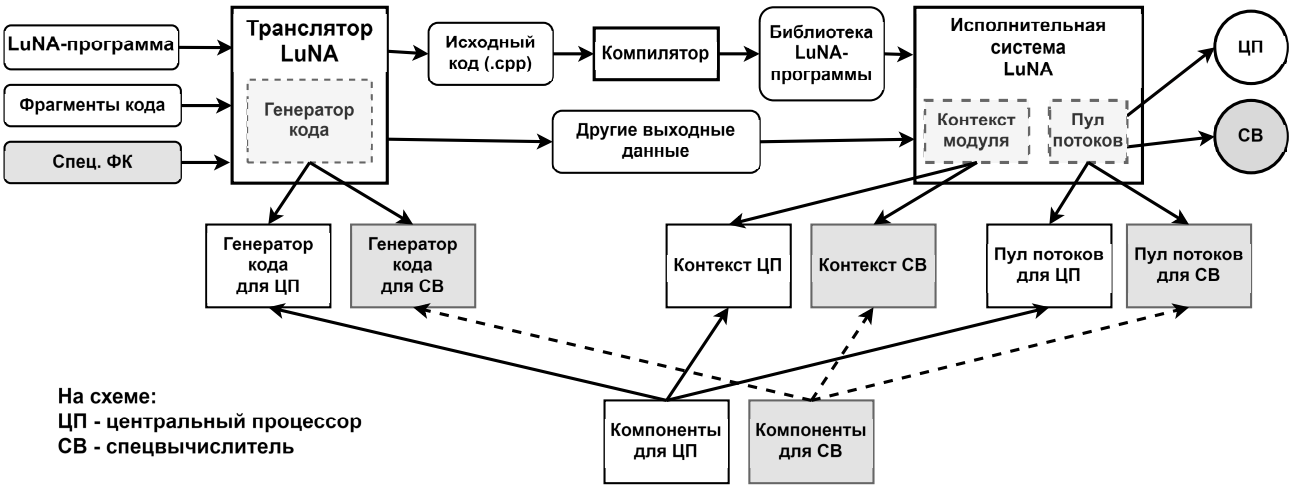


Рис. 4. Схема расширения архитектуры системы LuNA для поддержки спецвычислителей

```
05: opvar input t_floats X0 [-1] size=sX0
06: opvar input t_ints M0 [1] size=sM0
07: opvar output t_floats X implicit [sX0[0]] size=sX
08: opvar output t_floats X1 implicit [sX0[0]] size=sX1
09: rule float float int > float float
10:
11: __header__
12:     ...
13: __end__
14:
15: __before__
16:     ...
17: __end__
18:
```

```
19: __operator__
20:     __cpu__
21:         ...
22:     __end__
23:     __npu__
24:         ...
25:     __end__
26: __end__
27:
28: __operator__ BuiltinOperatorName
29: var input X implicit [sX0[0] / 2]
30: var input M0 explicit [1]
31: var output X1 implicit [sX0[0]]
32: rule float int float > complex64 int float
33: __start__
34:     ...
35: __end__
36:
37: __after__
38:     ...
39: __end__
```

В строках 1–9 расположена секция с метаинформацией. В строках 11–39 расположена секция с вычислениями. В метаинформации описываются такие компоненты, как название оператора, параметры и их допустимые типы, а также список комбинаций типов, которые будут использоваться в программе. Секция с вычислениями содержит четыре типа вычислений: вычисления перед запуском операторов на СВ, вычисления для собственного оператора, использование встроенных операторов и вычисления после запуска всех операторов на СВ.

Представленные результаты автоматизации использования СВ были частично опубликованы в работе [9].

3. Тестирование программ с использованием процессора Huawei Ascend. Тестирование программ, использующих процессор Ascend, проводилось на задачах блочного умножения плотных матриц и корреляционной свертки сейсмотрасс [10]. Целью тестирования выступает сравнение ручных C++-реализаций и LuNA-реализаций, а также сравнение использования центральных процессоров и процессоров Ascend для решения задач на предмет оценки качества при применении концепции активных знаний. Основной метрикой сравнения выступает время работы программ.

Запуски были выполнены на сервере Atlas 800 (модель 9000), установленном в ИВМиМГ СО РАН (см. описание на странице лаборатории ИИИТ [11]).

Задача блочного умножения плотных матриц. Цель тестирования этой задачи заключается в сравнении ручной реализации на языке C++ и автоматически конструируемых реализаций с использованием системы LuNA. Также сравнивается использование разного количества процессоров Ascend.

Для этой задачи матрицы разбиваются на блоки, и умножение двух блоков рассматривается как отдельная планируемая задача. В качестве сравнения выступают три реализации: 1) реализация на C++ только с вычислениями на процессоре Ascend (C++-

NPU реализация); 2) реализация на LuNA только с вычислениями на процессоре Ascend (LuNA-NPU реализация); 3) реализация на LuNA только с вычислениями на центральном процессоре (LuNA-CPU реализация).

Задача корреляционной свертки сейсмотрасс. Цель тестирования этой задачи заключается в применении концепции активных знаний для реальной прикладной задачи. Эта задача, как описано в [10], выполняет вычисление взаимнокорреляционной функции и быстрого преобразования Фурье (FFT). В качестве сравнения выступают три LuNA-реализации: 1) С использованием вычислений на центральном процессоре (при этом, эта реализация работает сопоставимо с ручной реализацией на C++). 2) С использованием собственного оператора процессора Ascend, в который были добавлены вычисления из предыдущего способа. 3) С использованием встроенных операторов процессора Ascend для вычисления FFT.

Результаты тестирования. На рис. 5, а, видно, что на больших размерах матриц использование процессора Ascend позволяет снизить время выполнения программы по сравнению с использованием центрального процессора.

На рис. 5, б, показано, что на больших размерах матриц использование большего количества процессоров Ascend позволяет снизить время выполнения программ в большей степени.

На рис. 5, в, показано сравнение реализаций C++-NPU и LuNA-NPU при использовании 4-х процессоров Ascend. По графику видно, что на больших размерах матриц ручная C++-реализация работает сопоставимо с LuNA-реализацией с точки зрения времени выполнения.

На рис. 5, г, видно, вопреки ожиданиям авторов, что вычисление FFT на процессоре Ascend значительно уступает вычислению FFT на центральном процессоре. При этом использование встроенных операторов для FFT уступает использованию собственного оператора.

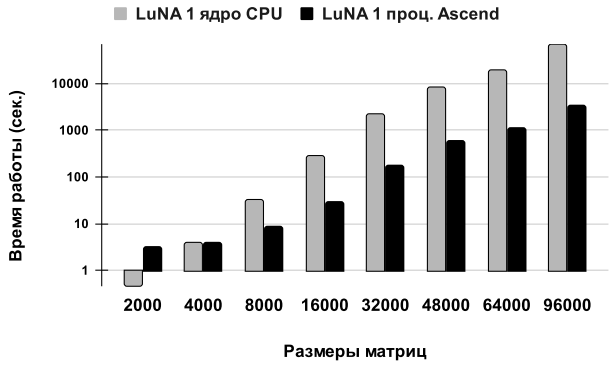
В целом можно отметить, что разработка LuNA-программ, использующих процессор Ascend, требует меньше трудозатрат, чем разработка ручных реализаций, при этом эти реализации работают сопоставимо. В большей степени этому способствует разработанный формат CoFaNA.

Заключение. В работе рассмотрен подход автоматического использования спецвычислителей на основе концепции активных знаний в системе LuNA. Было выполнено расширение архитектуры системы LuNA для поддержки различных спецвычислителей.

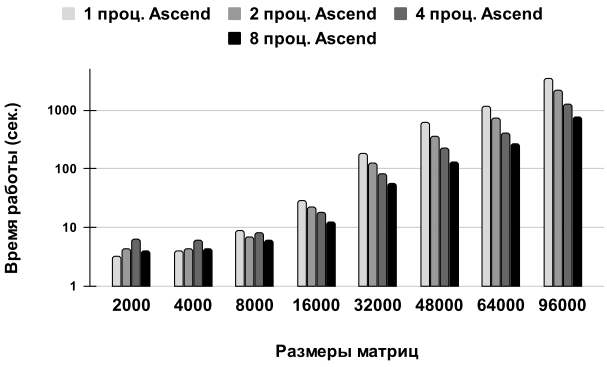
Реализована поддержка спецвычислителя Huawei Ascend в системе LuNA. Для упрощения работы с вычислениями на процессоре Huawei Ascend были предложены формат CoFaNA и пул потоков, поддерживающий такие вычисления.

На этом примере было показано, что на основе концепции активных знаний может быть обеспечено автоматическое конструирование программ с использованием спецвычислителей. При этом с пользователя снимается существенное количество работы, связанной с применением спецвычислителей и требуеющей детального знания их работы. Это существенно упрощает применение спецвычислителей.

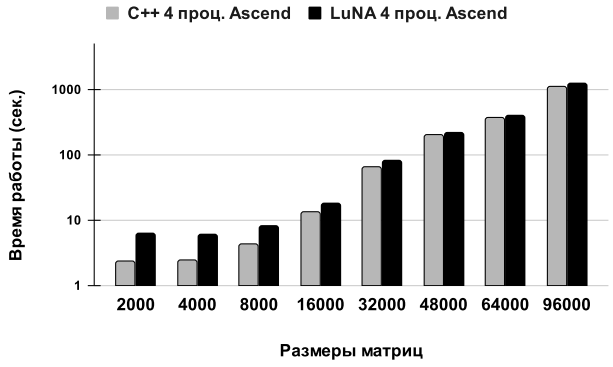
Результаты экспериментального исследования подтвердили, что использование процессора Huawei Ascend при решении задачи блочного умножения плотных матриц позволяет значительно повысить эффективность по сравнению с использованием только CPU. Основным результатом тестирования является то, что программы, построенные автоматически, по эффективности оказались близки к низкоуровневым ручным реализациям на



а)



б)



в)

	1 ядро ЦП	2 ядра ЦП
Центральный процессор (ЦП)	1.531 сек.	0.823 сек.
	1 проц. Ascend	2 проц. Ascend
Процессор Ascend (собств. оператор)	6.832 сек.	3.704 сек.
Процессор Ascend (встроен. операторы)	21.098 сек.	10.927 сек.

г)

Рис. 5. (а)–(в) — зависимость времени работы программ (логарифмическая шкала) от размеров матриц, (а) — LuNA-CPU на 1 ядре центр. проц. и LuNA-NPU на 1 проц. Ascend, (б) — LuNA-NPU на разном количестве проц. Ascend, (в) — C++-NPU и LuNA-NPU на 4-х проц. Ascend; (г) — зависимость времени работы алгоритма свертки от количества ресурсов и типа вычислений

C++. Это подтверждает применимость концепции активных знаний для решения таких задач автоматического конструирования программ.

Рассмотренный в работе подход на основе активных знаний является не единственным способом реализации поддержки спецвычислителей. Эта проблема исследуется и решается в разной степени в разных системах и компиляторах. В системе Mindspore [12], в которой все программы реализуются на языке Python, реализуется поддержка рассмотренного процессора Huawei Ascend для решения задач машинного обучения. Также поддержку Huawei Ascend реализует компилятор DPC++ [13]. В [14] приводится сравнение различных систем автоматического конструирования параллельных программ, обеспечивающих поддержку графических ускорителей (GPU): Charm++ [15], Legion [16], StarPU [17], StarSS [18].

В отличие от концепции активных знаний, которая позволяет автоматически конструировать эффективные программы в разных предметных областях (за счет разработки разных баз активных знаний), в представленных выше системах упор идет на автоматизацию конструирования эффективных программ ограниченного числа предметных областей (которые определяют область применения системы). Отдельно здесь стоит отметить систему Legion, которая позволяет эффективно работать с разными предметными областями, но с

требованием от пользователя проделывать некоторые этапы конструирования программы вручную.

В дальнейшем планируется реализовывать поддержку других спецвычислителей и при необходимости расширять подсистему поддержки спецвычислителей.

Список литературы

1. Malyshkin V. E. Active Knowledge, LuNA and Literacy for Oncoming Centuries // LNCS, T. 9465. 2015. P. 292–303. DOI: 10.1007/978-3-319-25527-9_19.

2. Malyshkin V. E., Perepelkin V. A. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem // Proceedings of the 11th International Conference on Parallel Computing Technologies (PaCT-2011), LNCS, 2011. T. 6873, P. 53–61. DOI: 10.1007/978-3-642-23178-0_5.

3. Малышкин В. Э., Перепелкин В. А. Построение баз активных знаний для автоматического конструирования решений прикладных задач на основе системы LuNA // Параллельные вычислительные технологии — XVIII всероссийская научная конференция с международным участием, ПаВТ'2024, г. Челябинск, 2–4 апреля 2024 г. Короткие статьи и описания плакатов. Челябинск: Издательский центр ЮУрГУ, 2024. С. 57–68. DOI: 10.14529/pct2024.

4. Liang X. Ascend AI Processor Architecture and Programming: Principles and Applications of CANN. Elsevier, 2020. DOI: 10.1016/C2020-0-00270-7.

5. Malyshkin V., Perepelkin V., Schukin G. Scalable Distributed Data Allocation in LuNA Fragmented Programming System // The Journal of Supercomputing, 2017. T. 73, Iss. 2, Springer. P. 726–732. DOI: 10.1007/s11227-016-1781-0.

6. Малышкин В. Э., Перепелкин В. А., Щукин Г. А. Распределенный алгоритм управления данными в системе фрагментированного программирования LuNA // Проблемы информатики, 2017. № 1(34). С. 74–88.

7. Using Operator Samples [Electron. Res.]: `https://www.hiascend.com/document/detail/zh/CANNCommunityEdition/82RC1alpha001/opdevg/tbeaicpudevg/atlasopdev_10_0025.html`.

8. AscendCL architecture and basic concepts [Electron. Res.]: `https://www.hiascend.com/document/detail/zh/CANNCommunityEdition/82RC1alpha001/appdevg/aclcppdevg/aclcppdevg_000004.html`.

9. Спирин В. А. Разработка алгоритмов автоматизации применения NPU в системе LuNA // Параллельные вычислительные технологии — XVIII всероссийская научная конференция с международным участием, ПаВТ'2024, г. Челябинск, 2–4 апреля 2024 г. Короткие статьи и описания плакатов. Челябинск: Издательский центр ЮУрГУ, 2024. С. 165-176. DOI: 10.14529/pct2024.

10. Хайретдинов М. С. Алгоритмы поточной свертки в задачах активного вибросейсмоакустического мониторинга / М. С. Хайретдинов, Г. М. Воскобойникова, Г. С. Седухина // Геосибирь, 2017.

11. Лаборатория искусственного интеллекта и информационных технологий [Электрон. Рес.]: `https://icmmg.nsc.ru/ru/content/pages/laboratoriya-iskusstvennogo-intellekta-i-informacionnyh-tehnologiy`.

12. Chen L. Deep learning and practice with mindspore. Springer Nature, 2021. DOI: 10.1007/978-981-16-2233-5.

13. Feng W., Maghareh R., Wang K. T. A. Extending DPC++ with Support for Huawei Ascend AI Chipset // International Workshop on OpenCL. 2021. P. 1–4. DOI: 10.1145/3456669.3456684.

14. Gu R., Becchi M. A comparative study of parallel programming frameworks for distributed GPU applications // Proceedings of the 16th ACM International Conference on Computing Frontiers. 2019. P. 268–273. DOI: 10.1145/3310273.3323071.

15. Robson M. P., Buch R., Kale L. V. Runtime coordinated heterogeneous tasks in Charm++ // 2016 Second International Workshop on Extreme Scale Programming Models and Middlewar (ESPM2). IEEE, 2016. P. 40–43. DOI: 10.1109/ESPM2.2016.011.

16. Bauer M. et al. Legion: Expressing locality and independence with logical regions // SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. IEEE, 2012. P. 1–11. DOI: 10.1109/SC.2012.71.

17. Augonnet C. et al. StarPU: a unified platform for task scheduling on heterogeneous multicore architectures // Euro-Par 2009 Parallel Processing: 15th International Euro-Par Conference, Delft, The Netherlands, August 25–28, 2009. Proceedings 15. Springer Berlin Heidelberg, 2009. P. 863–874. DOI: 10.1007/978-3-642-03869-3_80.

18. Ayguadé E. et al. An extension of the StarSs programming model for platforms with multiple GPUs // Euro-Par 2009 Parallel Processing: 15th International Euro-Par Conference, Delft, The Netherlands, August 25–28, 2009. Proceedings 15. Springer Berlin Heidelberg, 2009. P. 851–862. DOI: 10.1007/978-3-642-03869-3_79.

Виктор Эммануилович Малышкин — получил степень магистра математики в Томском государственном университете (1970), степень доктора технических наук в Новосибирском государственном университете (1993). В настоящее время является заведующим лабораторией синтеза параллельных программ в Институте вычислительной математики и математической геофизики СО РАН. Он также основал и в настоящее время возглавляет кафедру параллельных вычислений в Национальном исследовательском университете Новосибирска. Является одним из организаторов международной конференции PaCT (Parallel Computing Technologies), проводимых каждый нечетный год в России.

Имеет более 110 публикаций по параллельным и распределенным вычислениям, синтезу параллельных программ, суперкомпьютерному программному обеспечению и приложениям, параллельной реализации крупномасштабных численных моделей.

В настоящее время область его интересов включает параллельные вычислительные технологии, языки и системы параллельного программирования, методы и средства параллельной реализации крупномасштабных численных моделей, технологию активных знаний.

Victor Emmanuilovich Malyshkin graduated from Tomsk State University with the master of sciences degree in 1970, defended his candidate dissertation in physical and mathematical sciences in Computing Centre of SB RAS in 1984, and defended his doctoral dissertation in technical sciences in Novosibirsk State University (1993). Currently is head of parallel programs synthesis laboratory in the Institute of computational mathematics and mathematical geophysics SB RAS.

Is also a founder and head of the chair of parallel computing in Novosibirsk State University. Is one of organizers of the PaCT (Parallel Computing Technologies) international conference, being held each odd year in Russia. Has more than 110 publications on parallel and distributed computing, parallel programs synthesis, supercomputing software and applications, parallel implementation of large-scale numerical models. Current scientific interests include parallel computing technologies, languages and systems of parallel computing, methods and tools of software implementation of large-scale numerical models, the active knowledge technology.

Перепелкин Владислав Александрович — кандидат технических наук, старший научный сотрудник Института вычислительной математики и математической геофизики СО РАН; доцент кафедры параллельных вычислений факультета информационных технологий Новосибирского государственного университета. Тел.: (383) 330-89-94, e-mail: `perepelkin@ssd.sscc.ru`. В 2008 году

окончил Новосибирский государственный университет с присуждением степени магистра техники и технологии по направлению «Информатика и вычислительная техника». В 2023 году защитил кандидатскую диссертацию. Имеет более 20 опубликованных статей по теме автоматизации конструирования параллельных программ в области численного моделирования. Является одним из основных разработчиков экспериментальной системы автоматизации конструирования численных параллельных программ для мультикомпьютеров LuNA (от Language for Numerical Algorithms). Область профессиональных интересов включает автоматизацию конструирования параллельных программ, языки и системы параллельного программирования, высокопроизводительные вычисления.

Perepelkin Vladislav Aleksandrovich — PhD in Computer Science. Graduated from Novosibirsk State University in 2008 with the master degree in engineering and technology in computer science. Defended his PhD thesis in 2023. Nowadays has the senior researcher position at the Institute of Computational Mathematics and Mathematical Geophysics of the Siberian Branch of Russian Academy of Sciences, and also has a part time associated professor position in the Novosibirsk State University. He is an author of more than 20 papers on automation of numerical parallel programs construction. He is one of main developers of system LuNA (Language for Numerical Algorithms) for automatic construction of numerical parallel programs. Professional interests include automation of parallel programs construction, languages and systems of parallel programming, high performance computing.

Спирин Виталий Андреевич — студент магистратуры факультета информационных технологий Новосибирского государственного университета и инженер в Институте вычислительной математики и математической геофизики СО РАН. Получил диплом бакалавра по направлению «Информатика и вычислительная техника» в 2023 году. E-mail: `v.spirin@g.nsu.ru`. Область научных интересов: параллельные алгоритмы, высокопроизводительные вычисления, автоматическое конструирование программ.

Spirin Vitaly Andreevich is a master's student at the Faculty of Information Technology at Novosibirsk State University and an engineer at the Institute of Computational Mathematics and Mathematical Geophysics of the Siberian Branch of Russian Academy of Sciences. Graduated with a bachelor's degree in Computer Science in 2023. E-mail: `v.spirin@g.nsu.ru`. Research interests: parallel algorithms, high performance computing, automatic programs construction.

Дата поступления — 29.05.2025