AUTOMATED CONSTRUCTION OF HIGHLY-EFFICIENT PARALLEL PROGRAMS FOR SPARSE LINEAR ALGEBRA IN LUNA SYSTEM

N. Belyaev

Institute of Computational Mathematics and Mathematical Geophysics SB RAS,
630090, Novosibirsk, Russia

DOI: 10.24412/2073-0667-2022-3-46-60
EDN: MTTVTE

Automatic construction of efficient scientific parallel programs for supercomputers is in general a complex problem of system parallel programming. Therefore, various specialized system algorithms for automated parallel programs construction are of use. Of interest are algorithms for automated parallel program construction from a given numerical algorithm description. These algorithms can either be general purpose or specialized. The specialized parallel program construction algorithms are of particular interest. Despite the fact that such specialized system algorithms are applicable only for restricted classes of input numerical algorithms, such specialized algorithms have a significant advantage. These algorithms allow parallel program generators to employ parallel programming techniques which are widely used in manual parallel programming to construct highly efficient parallel programs. LuNA system for automatic construction of distributed parallel programs provides a basis for accumulation of such specialized system algorithms to provide high-quality parallel programs construction in particular subject domains. This system allows automated construction of parallel programs for a distributed memory parallel computer from a given numerical algorithm description written with high-level LuNA language. In this paper, a novel specialized algorithm of parallel programs construction for sparse linear algebra domain is presented. It is applicable for a class of sparse linear algebra algorithms which includes widely used preconditioner algorithms for sparse systems of linear equations. This fact makes the developed specialized system algorithm of practical interest. The presented specialized automated parallel program construction algorithm has been implemented as a specialized run-time system which has been integrated into the LuNA system and the module which has been integrated into the LuNA compiler. In order to integrate the implementation of the developed specialized system algorithm into the LuNA system the following approach is used. The LuNA compiler detects if the input numerical algorithm description written with LuNA language belongs to a class of numerical algorithms for which the LuNA system has a specialized support. In this case the corresponding specialized system algorithms are used to construct the so-called intermediate parallel program. Otherwise the previously developed general purpose parallel program construction algorithms are used. The control program constructs an executable representation of the input numerical algorithm at run-time. The executable representation of the input algorithm is a directed acyclic graph of single assignment variables and operations. This representation is then executed by the specialized run-time system. The specialized run-time system employs parallel programming techniques which are widely used in manual development of parallel preconditioners for sparse matrices. In order to assess the performance of parallel programs constructed using the developed specialized system algorithms, a test parallel program was automatically constructed from the description of sparse-vector multiply algorithm. The performance of the automatically constructed parallel program was compared to the

performance of implementation of the same algorithm from Intel MKL and Parallel Sparse BLAS libraries. Experimental results demonstrate that the performance of the automatically generated parallel program is comparable with the performance of the corresponding implementation from Intel MKL (the performance of the automatically constructed parallel program is only 15 % less than the performance of the matrix-vector multiply implementation from Intel MKL) . Also the constructed parallel program outperforms the implementation for Parallel Sparse BLAS library by approximately 10 %. These facts show that the LuNA system is practically applicable for automated construction of high performance distributed parallel programs for supercomputers in the sparse linear algebra field.

Key words: DAG, parallel preconditioner, parallel program synthesis, fragmented programming, task based parallelism, parallel run-time system, parallel programming system.

References

1. Schenk O., Gärtner K., Fichtner W. Efficient sparse LU factorization with left-right looking strategy on shared memory multiprocessors // BIT Numerical Mathematics. 2000. T. 40. N 1. P. 158–176.

2. Yamazaki I., Li X. S. New scheduling strategies and hybrid programming for a parallel right-looking sparse LU factorization algorithm on multicore cluster systems // 2012 IEEE 26th International Parallel and Distributed Processing Symposium. IEEE, 2012. P. 619–630.

3. Davis T., Stanley K. Sparse lu factorization of circuit simulation matrices // [Electron. Res.]: `www.cise.ufl.edu/davis/techreports/KLU/pp04.pdf`.

4. Geist G. A., Romine C. H. LU factorization algorithms on distributed-memory multiprocessor architectures // SIAM Journal on Scientific and Statistical Computing. 1988. T. 9. N 4. P. 639–649.

5. Choi J. et al. ScaLAPACK: A portable linear algebra library for distributed memory computers—Design issues and performance // Computer Physics Communications. 1996. T. 97. N 1–2. P. 1–15.

6. Choi J. et al. Design and implementation of the ScaLAPACK LU, QR, and Cholesky factorization routines //Scientific Programming. 1996. T. 5. N 3. P. 173–184.

7. Gates M. et al. Slate: Design of a modern distributed and accelerated linear algebra library // Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2019. P. 1–18.

8. Kurzak J. et al. Designing SLATE: Software for linear algebra targeting exascale. 2017.

9. Grigori L., Demmel J. W., Li X. S. Parallel symbolic factorization for sparse LU with static pivoting // SIAM Journal on Scientific Computing. 2007. T. 29. N 3. P. 1289–1314.

10. Schenk O. Scalable parallel sparse LU factorization methods on shared memory multiprocessors: dis. ETH Zurich, 2000.

11. Schenk O., Gärtner K., Fichtner W. Efficient sparse LU factorization with left-right looking strategy on shared memory multiprocessors // BIT Numerical Mathematics. 2000. T. 40. N 1. P. 158–176.

12. Toledo S. Locality of reference in LU decomposition with partial pivoting //SIAM Journal on Matrix Analysis and Applications. 1997. T. 18. N 4. P. 1065–1081.

13. Schenk O., Gärtner K. Solving unsymmetric sparse systems of linear equations with PARDISO // Future Generation Computer Systems. 2004. T. 20. N 3. P. 475–487.

14. Schenk O. et al. PARDISO: a high-performance serial and parallel sparse linear solver in semiconductor device simulation // Future Generation Computer Systems. 2001. T. 18. N 1. P. 69–78.

15. Amestoy P. R. et al. MUMPS: a general purpose distributed memory sparse solver // International Workshop on Applied Parallel Computing. Springer, Berlin, Heidelberg, 2000. P. 121–130.

16. Hysom D., Pothen A. A scalable parallel algorithm for incomplete factor preconditioning // SIAM Journal on Scientific Computing. 2001. T. 22. N 6. P. 2194–2215.

17. Hysom D., Pothen A. Efficient parallel computation of ILU (k) preconditioners // Proceedings of the 1999 ACM/IEEE conference on Supercomputing. 1999. P. 29-es.

18. Kale L. V., Krishnan S. Charm++ A portable concurrent object oriented system based on C++ // Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications. 1993. P. 91–108.

19. 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.

20. Slaughter E. et al. Regent: A high-productivity programming language for HPC with logical regions // Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2015. P. 1–12.

21. Malyshkin V. E., Perepelkin V. A. LuNA fragmented programming system, main functions and peculiarities of run-time subsystem // International Conference on Parallel Computing Technologies. Springer, Berlin, Heidelberg, 2011. P. 53–61.

22. Belyaev N., Perepelkin V. High-Efficiency Specialized Support for Dense Linear Algebra Arithmetic in LuNA System // International Conference on Parallel Computing Technologies. Springer, Cham, 2021. P. 143–150.

23. Bosilca G. et al. Parsec: Exploiting heterogeneity to enhance scalability //Computing in Science & Engineering. 2013. T. 15. N 6. P. 36–45.

24. Malyshkin V. E. Tekhnologiya fragmentirovannogo programmirovaniya // Vestnik YUzhno-Ural'skogo gosudarstvennogo universiteta. Seriya: Vychislitel'naya matematika i informatika. 2012. N 46 (305).

25. Bichot C. E., Siarry P. (ed.). Graph partitioning. John Wiley & Sons, 2013.

26. Pellegrini F. Scotch and libScotch 5.1 user's guide. 2008.

27. Karypis G., Kumar V. METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. 1997.

28. ZHdanov A. I., Bogdanova E. YU. Ob odnoj vychislitel'noj realizacii blochnogo metoda Gaussa-Zejdelya dlya normal'nyh sistem uravnenij // Vestnik Samarskogo gosudarstvennogo tekhnicheskogo universiteta. Seriya Fiziko-matematicheskie nauki. 2016. T. 20. N 4.

29. Davis T. A., Hu Y. The University of Florida sparse matrix collection //ACM Transactions on Mathematical Software (TOMS). 2011. T. 38. N 1. P. 1–25.

30. Wang E. et al. Intel math kernel library // High-Performance Computing on the Intel§Xeon Phi^{TM}. Springer, Cham, 2014. P. 167–188.

31. [Electron. Res.]: https://www.jscc.ru/

32. Filippone S., Colajanni M. PSBLAS: A library for parallel linear algebra computation on sparse matrices // ACM Transactions on Mathematical Software (TOMS). 2000. T. 26. N 4. P. 527–550.

АВТОМАТИЧЕСКОЕ КОНСТРУИРОВАНИЕ ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ ДЛЯ ЗАДАЧ РАЗРЕЖЕННОЙ ЛИНЕЙНОЙ АЛГЕБРЫ В СИСТЕМЕ LUNA

Н. А. Беляев

Институт вычислительной математики и математической геофизики СО РАН,
630090, Новосибирск, Россия

УДК 004.4'242
DOI: 10.24412/2073-0667-2022-3-46-60
EDN: MTTVTE

В статье описываются разработанные специализированные системные алгоритмы автоматического конструирования параллельной программы по описанию численного алгоритма для задач разреженной линейной алгебры. Разработанные специализированные системные алгоритмы позволяют применять при автоматическом конструировании параллельных программ техники параллельного программирования, которые широко применяются при ручной разработке параллельных программ в данной предметной области. Разработанные системные алгоритмы автоматического конструирования параллельной программы были реализованы в виде модулей, которые были интегрированы в систему параллельного программирования общего назначения LuNA, разрабатываемую в ИВМ и МГ СО РАН. Производительность тестовых параллельных программ, автоматически сконструированных системой LuNA, оказалась сравнима с производительностью популярных широко используемых библиотечных реализаций этих алгоритмов разреженной линейной алгебры.

Ключевые слова: граф задач, параллельная реализация предобуславливателя, конструирование параллельной программы, параллелизм уровня задач.

Введение. Проблема разработки средств автоматического конструирования параллельной программы по описанию численного алгоритма является актуальной проблемой. В частности, сложность представляет разработка алгоритмов автоматического конструирования параллельной программы, эффективно использующей доступные ресурсы вычислителя. Такие программы целесообразно автоматически конструировать по описанию численного алгоритма, применяя специализированные системные алгоритмы автоматического конструирования параллельной программы вместо универсальных.

Специализированные алгоритмы позволяют применять при автоматическом конструировании параллельной программы широко известные и отработанные техники параллельного программирования, применяемые при ручной разработке параллельных программ в конкретной области численного моделирования. Показательным примером таких техник параллельного программирования является техника right-looking [1–3], применяемая при разработке программ, реализующих алгоритм LU разложения плотной матрицы [4] для параллельного компьютера с распределенной памятью. Многие библиотечные реализации LU разложения разработаны с применением этой техники [5–8]. Применение таких техник

во многом обуславливает высокую эффективность рассмотренных библиотечных параллельных программ. Достигать высокой эффективности автоматически сконструированной параллельной программы в этих обстоятельствах целесообразно путем применения при конструировании специализированных системных алгоритмов, реализующих такие техники.

На сегодняшний день опубликовано большое количество работ, посвященных разработке и применению различных узкоспециализированных техник параллельного программирования для задач численного моделирования. Так, в работах [9, 10] предлагаются различные реализации LU-разложения разреженной матрицы, основанные на применении техник параллельного программирования right-looking и left-looking [11, 12]. Различные параллельные библиотечные реализации [13–15] алгоритма LL^T-разложения разреженных матриц также используют опубликованные широко известные практики, в частности, технику left-looking. Аналогичная ситуация наблюдается и для другого класса численных алгоритмов — предобуславливателей разреженных СЛАУ [16–17].

Также активно разрабатываются автоматические средства параллельного программирования как общего назначения, так и специализированные. В частности, Charm++ [18] позволяет автоматически конструировать параллельную программу по описанию численного алгоритма в виде множества взаимодействующих легковесных процессов. Системы Legion [19], Regent [20] и LuNA [21] автоматически конструируют параллельную программу по описанию численного алгоритма в виде двудольного бесконтурного орграфа переменных и операций. Эти системы используют универсальные алгоритмы распределенного исполнения такого графа на параллельном компьютере с распределенной памятью, а также возможна интеграция в них модулей, реализующих специализированные системные алгоритмы конструирования параллельной программы по описанию численного алгоритма. Численные алгоритмы можно разделить на классы согласно тому, какие техники параллельного программирования применимы при параллельной реализации конкретных численных алгоритмов. Для каждого класса в перечисленные системы параллельного программирования возможно интегрировать отдельные модули, реализующие специализированные алгоритмы автоматического конструирования параллельной программы. Таким образом, возможно обеспечить накопление реализаций специализированных алгоритмов автоматического конструирования параллельной программы по описанию численного алгоритма в системах параллельного программирования общего назначения.

В частности, ранее с участием автора в работе [22] был предложен набор специализированных системных алгоритмов автоматического конструирования параллельной программы для класса численных алгоритмов факторизации плотной матрицы, включающего алгоритмы LU разложения, LL^T разложения и пр. Разработанные специализированные алгоритмы конструирования были реализованы в виде модулей, которые были интегрированы в систему параллельного программирования LuNA. Это позволило автоматически сконструировать высокоэффективную параллельную программу для параллельного компьютера с распределенной памятью по описанию алгоритма LL^T разложения плотной матрицы. Производительность автоматически сконструированной параллельной программы превысила производительность реализации LL^T разложения, взятой из широко применяемой библиотеки ScaLAPACK. Это показало практическую значимость подхода накопления в системе параллельного программирования общего назначения специализированных системных алгоритмов конструирования параллельной программы и пригодность системы LuNA для такого накопления.

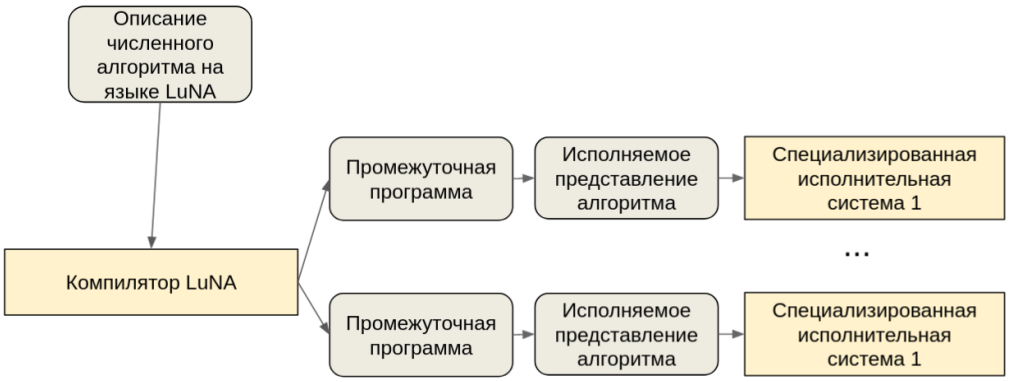

Рис. 1. Применяемая схема конструирования параллельной программы

Структурно дальнейшее изложение организовано следующим образом. В разделе 1 дается краткое описание системы LuNA, описываются основные компоненты системы и объясняется их назначение, а также объясняется предлагаемая схема конструирования параллельной программы. Далее объясняется идея, стоящая за реализованной схемой автоматического конструирования параллельной программы. Затем описывается представление численного алгоритма, по которому разработанными системными алгоритмами конструируется параллельная программа. Далее описываются разработанные системные алгоритмы конструирования параллельной программы и технические детали их реализации. В конце статьи приводятся результаты измерения производительности параллельных программ, сконструированных с помощью разработанных системных алгоритмов.

1. Применяемая схема конструирования параллельной программы. Рассмотрим применяемую схему конструирования параллельной программы. Эта схема предлагалась ранее в работе [22] и поэтому применена и в данной работе. Опишем ее.

Компилятор системы LuNA, произведя лексический и синтаксический анализ текста входного представления численного алгоритма, определяет, принадлежит ли входное описание численного алгоритма к одному из классов численных алгоритмов, для которых в системе LuNA имеется специализированная поддержка. В этом случае компилятором конструируется параллельная программа с помощью соответствующего специализированного алгоритма конструирования, реализованного в виде модуля компилятора. Эта сконструированная параллельная программа, в свою очередь, конструирует **исполняемое** представление исходного численного алгоритма и подает его на вход специализированной исполнительной системе, которая исполняет его на параллельном компьютере с распределенной памятью. Подход использования такой промежуточной программы применяется при разработке специализированных систем параллельного программирования, в частности, [23]. Если же входное описание численного алгоритма не принадлежит ни к одному из классов, для которых в системе LuNA имеется специализированная поддержка, то для конструирования параллельной программы используется исполнительная система общего назначения. Схема проиллюстрирована на рис. 1.

Применение описанной схемы позволяет реализовать в специализированных исполнительных системах системные алгоритмы конструирования параллельной программы с использованием некоторых отработанных техник параллельного программирования, которые широко применяются при ручной разработке параллельных программ в конкретной предметной области.

2. Представление численного алгоритма. Исполняемое представление численного алгоритма было разработано на основе ранее разработанного т. н. фрагментированного представления алгоритма [24].

Численный алгоритм представляется в виде конечного двудольного бесконтурного орграфа, вершинами которого являются **переменные** единственного присваивания и **операции** единственного срабатывания, а дуги графа обозначают информационные зависимости между операциями. Множество допустимых значений каждой конкретной переменной определяется разработчиком описания численного алгоритма (например, значением переменной может выступать блок матрицы, вектор, целое число и т. д). Будем называть переменную **входной** переменной операции, если от этой переменной к операции идет дуга. Будем называть переменную **выходной** переменной операции, если от операции к этой переменной идет дуга. Если переменная является входной переменной некоторой операции, будем называть такую операцию **операцией-потребителем** этой переменной. В случае, если переменная является вершиной-источником в графе, будем называть ее **входной** переменной алгоритма, а если переменная является вершиной-стоком, то — **выходной** переменной алгоритма. Каждой операции поставлен в соответствие модуль, реализующий некоторую функцию вычисления значений выходных переменных операции по значениям входных переменных. Такие модули предоставляются разработчиком описания численного алгоритма. Примером модуля может служить процедура, реализованная на процедурном языке (C, C++, Fortran).

Пусть на некоторых непересекающихся подмножествах переменных определены функции, по одной на подмножество, называемые в дальнейшем **индексными функциями**. Индексная функция переменной ставит в соответствие кортеж целых неотрицательных чисел фиксированной длины для каждой конкретной функции. Элементы таких кортежей будем называть **индексами** переменной. В случае, если для какого-то подмножества переменных определена индексная функция, будем говорить, что это подмножество переменных объединено в ***n*-мерный массив**, где n — размер соответствующих кортежей индексов. Переменные, объединенные в массив, будем называть **элементами** массива.

Рассмотрим теперь исполнение описанного представления численного алгоритма, для этого сначала рассмотрим исполнение операции. Исполнение операции заключается в исполнении модуля, поставленного в соответствие операции. В результате исполнения операции вычисляются значения ее выходных переменных. Эти переменные будем считать вычисленными. Будем также считать вычисленными входные переменные алгоритма. Операция может быть исполнена в случае, если у нее либо отсутствуют входные переменные, либо все ее входные переменные вычислены (такие операции будем называть **готовыми к исполнению**). Исполнение данного представления алгоритма — это последовательность шагов, на каждом из которых исполняются все готовые операции. После исполнения всех готовых операций на каждом шаге вычисляются значения некоторого (возможно, пустого) множества переменных. В результате получается новое множество (возможно, пустое) готовых операций, которые также исполняются. Такой процесс продолжается до тех пор, пока на очередном шаге множество готовых операций не окажется пустым.

Описанное выше исполняемое представление численного алгоритма будем называть **графовым** представлением алгоритма.

3. Описание поддерживаемого класса численных алгоритмов. Опишем теперь класс численных алгоритмов, для которых применимы разработанные специализирован-

ные алгоритмы конструирования параллельной программы. Для описания этого класса сформулируем критерии, которым должен удовлетворять численный алгоритм:

— Численный алгоритм представим в виде, описанном в разделе 2 (поскольку это представление является входным для разработанных алгоритмов конструирования параллельной программы).

— Для каждой операции в графовом представлении алгоритма верно, что среди ее выходных переменных найдется единственный элемент массива, причем этот массив одномерный.

— В графовом представлении алгоритма максимальная размерность массива, в который объединены переменные, не превышает двух.

Данные критерии являются достаточными для того, чтобы графовое представление численного алгоритма можно было исполнить с помощью описанных ниже разработанных системных алгоритмов. Описанным критериям удовлетворяют в т. ч. многие алгоритмы линейной алгебры, такие как алгоритмы разреженного блочного матрично-векторного умножения, блочные алгоритмы решения треугольных разреженных СЛАУ методом Гаусса, а также ряд алгоритмов предобуславливателей разреженных СЛАУ, таких как предобуславливатель Гаусса-Зейделя. Эти алгоритмы широко используются при разработке итерационных решателей разреженных СЛАУ, что обусловливает практическую значимость выбранного класса численных алгоритмов. Описанным критериям удовлетворяют и другие численные алгоритмы.

4. Алгоритм конструирования параллельной программы. Разработанный алгоритм исполнения графового представления численного алгоритма можно разбить на две основные стадии: построение распределения операций и переменных по узлам параллельного компьютера с распределенной памятью и исполнение на каждом из узлов параллельного компьютера отображенного на него подграфа переменных и операций.

Построение распределения операций и переменных графового представления по узлам параллельного компьютера с распределенной памятью заключается в выполнении следующих шагов. Сначала по графовому представлению строится граф операций. Построение его состоит в том, что из первоначального графа удаляются вершины-переменные. При этом в результирующем графе дуга идет от одной операции к другой в случае, если в исходном графе существовал простой путь длины 2 от одной операции к другой. Пример преобразования исходного графа переменных и операций в граф операций показан на рис. 2.

Кружками обозначены переменные, а прямоугольниками — операции. Слева показан пример некоторого исходного графа переменных и операций, а справа — построенный по нему граф операций.

Следующим шагом осуществляется построение распределения вершин графа операций по узлам параллельного компьютера с распределенной памятью. Для этого граф операций разделяется на k подграфов по числу узлов параллельного компьютера с распределенной памятью так, что одна вершина графа операций принадлежит единственному подграфу. При этом минимизируются два показателя. Во-первых, для каждой пары подграфов минимизируется суммарное количество дуг, соединяющих вершины, принадлежащие этим подграфам. Во-вторых, минимизируется максимальная разность числа узлов для любых двух подграфов. Эта задача широко известна [25], известны эффективные библиотечные реализации алгоритмов решения этой задачи [26, 27]. В данной работе для решения этой задачи применяется соответствующий модуль библиотеки METIS [28]. Каждый подграф

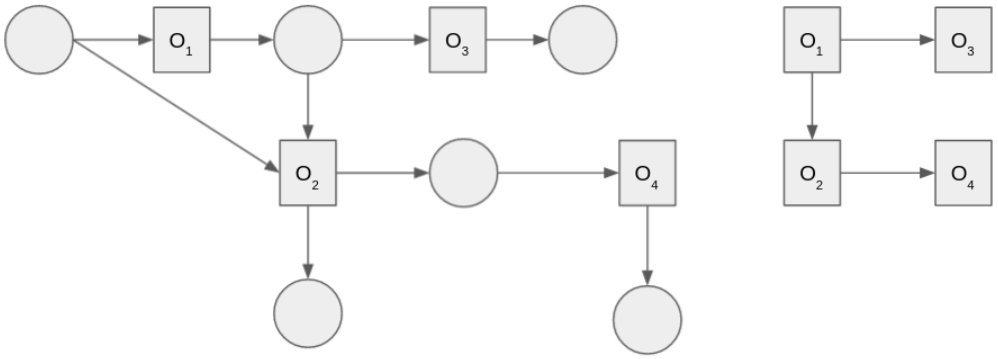

Рис. 2. Пример построения графа операций по исходному графу переменных и операций

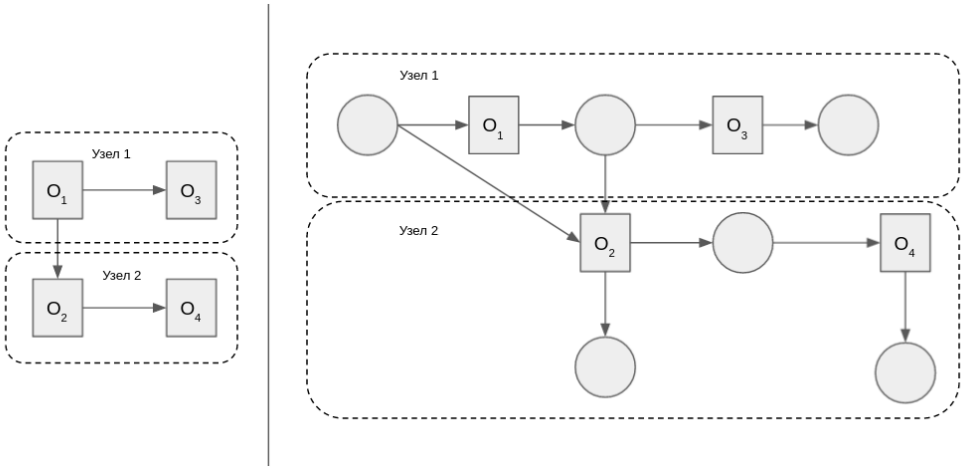

Рис. 3. Пример распределения графа переменных и операций по узлам параллельного компьютера с распределенной памятью

считается отображаемым на отдельный узел параллельного компьютера. Далее строится распределение переменных по узлам параллельного компьютера. Каждая переменная отображается на тот узел параллельного компьютера, на который отображена операция, вычисляющая значение этой переменной. Пример распределения исходного графа переменных и операций на узлы параллельного компьютера с распределенной памятью показан на рис. 3.

Кружками обозначены переменные, а прямоугольниками — операции. Слева показан пример разбиения на k частей графа операций, справа показано распределение переменных и операций исходного графа по узлам параллельного компьютера с распределенной памятью, построенное по данному разбиению графа операций. Пунктирной линией выделены подграфы, отображенные на разные узлы параллельного компьютера с распределенной памятью.

Исполнение подграфа переменных и операций, отображенного на узел параллельного компьютера, происходит согласно следующему алгоритму:

— Осуществляется поиск операций, для которых выполнены два условия: операция готова и значения всех входных переменных операции находятся в памяти того же узла параллельного компьютера, на который отображена операция. Такие операции вместе

со значениями своих входных переменных формируют вычислительные задачи, которые помещаются в очередь готовых для исполнения задач.

— Вычислительные задачи из очереди готовых задач исполняются, исполнение задачи заключается в исполнении соответствующей операции. Для каждой вычисленной переменной осуществляется поиск ее операций-потребителей, при этом, если такая операция отображена на другой узел параллельного компьютера, осуществляется асинхронная пересылка значения переменной на нужный узел с использованием коммуникационной сети параллельного компьютера.

— Шаги 1–3 повторяются до тех пор, пока на шаге 1 множество готовых операций не окажется пустым.

5. Тестирование. Практическую значимость разработанных системных алгоритмов автоматического конструирования параллельных программ обусловливает производительность конструируемых программ. Производительность конструируемых параллельных программ, таким образом, требуется оценить. Это целесообразно сделать путем сравнения на идентичном наборе входных данных производительности автоматически сконструированной параллельной программы и параллельной программы, разработанной вручную экспертами конкретной предметной области и реализующей тот же численный алгоритм. Далее описываются параметры проведения и результаты такого сравнения.

Тестовые параллельные программы были сконструированы по описанию численных алгоритмов разреженного блочного матрично-векторного умножения и блочного предобуславливателя Гаусса-Зейделя [28] раразеженной СЛАУ. Выбор алгоритмов обусловлен тем, что эти алгоритмы широко применяются при разработке итерационных решателей разреженных СЛАУ, а также доступны реализации этих алгоритмов в составе широко применяемых в данной предметной области библиотек. В качестве входных данных выступала квадратная матрица „Queen“ из общедоступного онлайн хранилища матриц Florida Matrix Colletction [29]. Матрица выбрана исходя из доступного объема оперативной памяти на узлах использованного параллельного компьютера, а хранилище матриц Florida Matrix Collection выбрано в качестве источника входных данных, поскольку в нем представлены матрицы, получаемые при решении реальных задач численного моделирования и являющиеся показательными примерами входных данных для различных алгоритмов решения СЛАУ. В качестве вектора для задачи матрично-векторного умножения и входного вектора для предобуславливателя Гаусса-Зейделя выступал единичный вектор соответствующего размера (выбранные значения векторов обеспечивают корректную работу тестовых программ, при этом эти значения не оказывают влияния на их производительность). Входная матрица для сконструированных системой LuNA программ была разделена на квадратные разреженные блоки, представленные в формате CSR. В качестве библиотеки модулей, реализующих операции, использовавшиеся в описаниях алгоритмов для системы LuNA (умножения блока входной матрицы на вектор и решения треугольной СЛАУ с блоком входной матрицы), выступала библиотека Intel MKL [30] версии 2021.4.0. В качестве вычислителя использовался суперкомпьютер МВС-10П [31] в составе межведомственного суперкомпьютерного центра Российской академии наук. Узел суперкомпьютера имеет в своем составе два процессора Intel Xeon E5-2690 (каждый имеет 8 ядер) и 16 GB оперативной памяти. Производительность параллельных программ, сконструированных системой LuNA, измерялась как в случае запуска на единственном узле суперкомпьютера, так и в случае запуска на множестве узлов, поскольку разработанные системные алгоритмы кон-

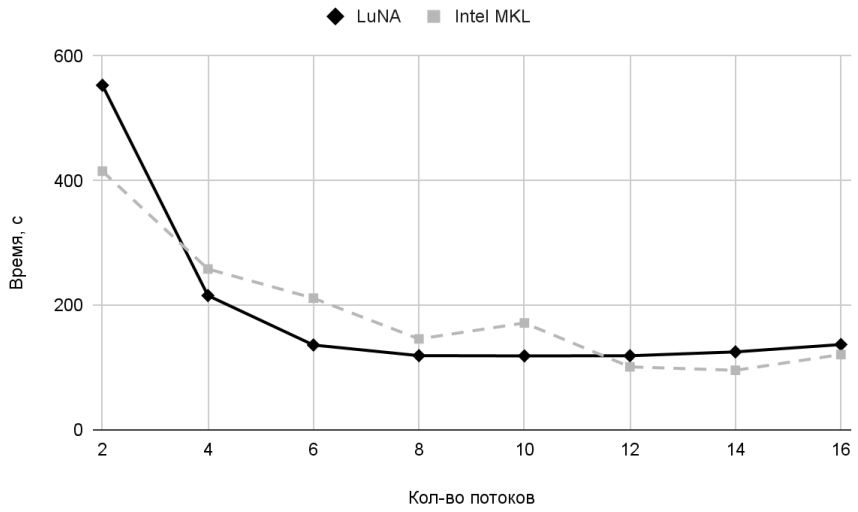

Рис. 4. Результаты измерения производительности параллельной программы, сконструированной системой LuNA, и реализации алгоритма разреженного матрично-векторного умножения из библиотеки Intel MKL

струирования параллельной программы пригодны и для конструирования программы для параллельного компьютера с общей памятью.

Тест 1. Производительность автоматически сконструированной параллельной программы, реализующей алгоритм блочного матрично-векторного умножения, сравнивалась с реализацией алгоритма разреженного матрично-векторного умножения в составе библиотеки Intel MKL. Обеими паралельными программами производилось по одной тысяче умножений разреженной матрицы на вектор для того чтобы имитировать итерационный процесс, присутствующий в алгоритмах итерационного решения разреженных СЛАУ. В качестве вычислителя использовался единственный узел суперкомпьютера МВС-10П. Количество потоков, использовавшихся обеими параллельными программами, варьировалось от одного до шестнадцати по числу доступных на узле ядер процессора. Результаты измерения производительности приводятся на рис. 4.

Производительность автоматически сконструированной параллельной программы уступает производительности библиотечной реализации незначительно (на 15–20 %), при этом система LuNA конструирует программу автоматически по описанию численного алгоритма. Незначительная разница в производительности не является критичной, поскольку ручная разработка параллельной программы, реализующей блочный алгоритм разреженного матрично-векторного умножения со сравнимой производительностью, требует значительно больших временных затрат, чем разработка описания численного алгоритма для системы LuNA.

Тест 2. По описанию алгоритма разреженного матрично-векторного умножения системой LuNA была автоматически сконструирована параллельная программа. Производительность автоматически сконструированной параллельной программы сравнивалась с реализацией алгоритма разреженного матрично-векторного умножения из библиотеки Parallel Sparse BLAS [32]. Аналогично тесту 1, обеими программами производилось по одной тысяче матрично-векторных умножений. Количество используемых при запуске обеих программ узлов суперкомпьютера варьировалось от одного до шестнадцати. На каждом

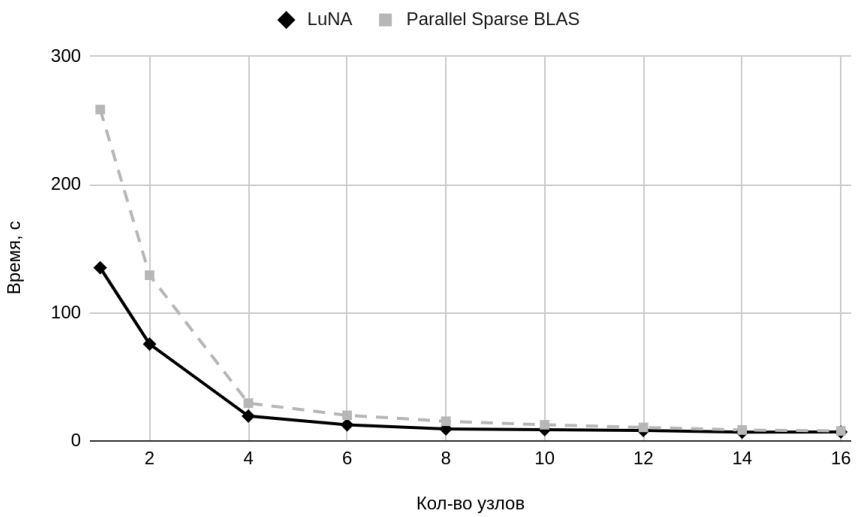

Рис. 5. Результаты измерения производительности параллельной программы, сконструированной системой LuNA, и реализации алгоритма разреженного матрично-векторного умножения из библиотеки Parallel Sparse BLAS

узле запускалось по 32 потока по числу доступных процессорных ядер. Результаты измерения производительности программ приведены на рис. 5.

Производительность параллельной программы, автоматически сконструированной системой LuNA, превосходит производительность реализации разреженного матрично-векторного умножения из библиотеки Parallel Sparse BLAS для параллельного компьютера с распределенной памятью, что также говорит в пользу практической значимости разработанных системных алгоритмов автоматического конструирования параллельной программы.

Тест 3. По описанию алгоритма предобуславливателя Гаусса-Зейделя для разреженной СЛАУ с помощью системы LuNA сконструирована параллельная программа. Тест демонстрирует работоспособность разработанных системных алгоритмов конструирования параллельной программы на популярном численном алгоритме из поддерживаемого системой LuNA класса алгоритмов предобуславливателей разреженных СЛАУ. Аналогично предыдущим тестам, сконструированной параллельной программой производилась одна тысяча применений предобуславливателя к исходному вектору. Количество используемых при запуске обеих программ узлов суперкомпьютера варьировалось от одного до четырех. На каждом узле запускалось по 32 потока по числу доступных процессорных ядер. Результаты измерения производительности автоматически сконструированной программы приведены на рис. 6.

Разработанные специализированные алгоритмы конструирования параллельной программы показали работоспособность на примере численного алгоритма предобуславливателя Гаусса-Зейделя, который выступает в качестве демонстрационного примера алгоритма предобуславливателя разреженной СЛАУ. Эффективность сконструированной с помощью разработанных специализированных алгоритмов параллельной программы близка к 90 процентам, что также является приемлемым в данной предметной области и подтверждает практическую значимость полученных результатов.

Заключение. В статье предложена схема автоматического конструирования параллельной программы, основанная на накоплении в системе параллельного программирова-

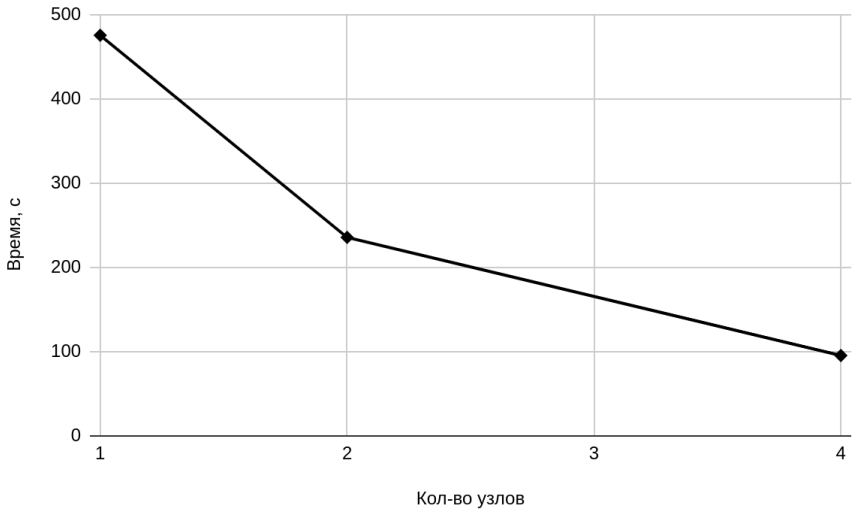

Рис. 6. Результаты измерения производительности параллельной программы, сконструированной системой LuNA

ния общего назначения специализированных системных алгоритмов конструирования программы. Для класса численных алгоритмов разреженной линейной алгебры разработаны специализированные системные алгоритмы конструирования параллельной программы. Разработанные алгоритмы были реализованы в виде модулей, которые были интегрированы в систему параллельного программирования общего назначения LuNA.

Производительность параллельных программ, конструируемых с помощью интегрированных в систему LuNA модулей, сравнивалась с производительностью широко используемых реализаций соответствующих численных алгоритмов, взятых из библиотек Intel MKL и Parallel Sparse BLAS. Производительность сконструированных программ незначительно уступала производительности библиотечных реализаций соответствующих алгоритмов из библиотеки Intel MKL и превосходила производительность реализаций из библиотеки Parallel Sparse BLAS. Этот факт говорит в пользу практической значимости разработанных специализированных системных алгоритмов автоматического конструирования параллельной программы.

Несмотря на то, что проблема автоматического конструирования параллельных программ остается в общем случае нерешенной, разработанные системные алгоритмы открывают возможности дальнейшего усовершенствования системы LuNA в части расширения класса алгоритмов, для которых в системе имеется специализированная поддержка. В частности, открываются возможности по добавлению в систему LuNA специализированной поддержки для практически значимого класса алгоритмов итерационного решения разреженной СЛАУ с предобуславливанием. При этом разработанные системные алгоритмы окажутся применимыми для конструирования высокоэффективных параллельных подпрограмм, реализующих алгоритмы предобуславливания СЛАУ.

В дальнейшем планируется разработать и реализовать специализированные системные алгоритмы автоматического конструирования параллельной программы по описанию численного алгоритма итерационного решения разреженной СЛАУ с предобуславливанием.

Список литературы

1. Schenk O., Gärtner K., Fichtner W. Efficient sparse LU factorization with left-right looking strategy on shared memory multiprocessors // BIT Numerical Mathematics. 2000. T. 40. N 1. P. 158–176.

2. Yamazaki I., Li X. S. New scheduling strategies and hybrid programming for a parallel right-looking sparse LU factorization algorithm on multicore cluster systems // 2012 IEEE 26th International Parallel and Distributed Processing Symposium. IEEE, 2012. P. 619–630.

3. Davis T., Stanley K. Sparse lu factorization of circuit simulation matrices // [Electron. Res.]: `www.cise.ufl.edu/davis/techreports/KLU/pp04.pdf`.

4. Geist G. A., Romine C. H. LU factorization algorithms on distributed-memory multiprocessor architectures // SIAM Journal on Scientific and Statistical Computing. 1988. T. 9. N 4. P. 639–649.

5. Choi J. et al. ScaLAPACK: A portable linear algebra library for distributed memory computers—Design issues and performance // Computer Physics Communications. 1996. T. 97. N 1–2. P. 1–15.

6. Choi J. et al. Design and implementation of the ScaLAPACK LU, QR, and Cholesky factorization routines //Scientific Programming. 1996. T. 5. N 3. P. 173–184.

7. Gates M. et al. Slate: Design of a modern distributed and accelerated linear algebra library // Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2019. P. 1–18.

8. Kurzak J. et al. Designing SLATE: Software for linear algebra targeting exascale. 2017.

9. Grigori L., Demmel J. W., Li X. S. Parallel symbolic factorization for sparse LU with static pivoting // SIAM Journal on Scientific Computing. 2007. T. 29. N 3. P. 1289–1314.

10. Schenk O. Scalable parallel sparse LU factorization methods on shared memory multiprocessors: dis. ETH Zurich, 2000.

11. Schenk O., Gärtner K., Fichtner W. Efficient sparse LU factorization with left-right looking strategy on shared memory multiprocessors // BIT Numerical Mathematics. 2000. T. 40. N 1. P. 158–176.

12. Toledo S. Locality of reference in LU decomposition with partial pivoting //SIAM Journal on Matrix Analysis and Applications. 1997. T. 18. N 4. P. 1065–1081.

13. Schenk O., Gärtner K. Solving unsymmetric sparse systems of linear equations with PARDISO // Future Generation Computer Systems. 2004. T. 20. N 3. P. 475–487.

14. Schenk O. et al. PARDISO: a high-performance serial and parallel sparse linear solver in semiconductor device simulation // Future Generation Computer Systems. 2001. T. 18. N 1. P. 69–78.

15. Amestoy P. R. et al. MUMPS: a general purpose distributed memory sparse solver // International Workshop on Applied Parallel Computing. Springer, Berlin, Heidelberg, 2000. P. 121–130.

16. Hysom D., Pothen A. A scalable parallel algorithm for incomplete factor preconditioning // SIAM Journal on Scientific Computing. 2001. T. 22. N 6. P. 2194–2215.

17. Hysom D., Pothen A. Efficient parallel computation of ILU (k) preconditioners // Proceedings of the 1999 ACM/IEEE conference on Supercomputing. 1999. P. 29-es.

18. Kale L. V., Krishnan S. Charm++ A portable concurrent object oriented system based on C++ // Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications. 1993. P. 91–108.

19. 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.

20. Slaughter E. et al. Regent: A high-productivity programming language for HPC with logical regions // Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2015. P. 1–12.

21. Malyshkin V. E., Perepelkin V. A. LuNA fragmented programming system, main functions and peculiarities of run-time subsystem // International Conference on Parallel Computing Technologies. Springer, Berlin, Heidelberg, 2011. P. 53–61.

22. Belyaev N., Perepelkin V. High-Efficiency Specialized Support for Dense Linear Algebra Arithmetic in LuNA System // International Conference on Parallel Computing Technologies. Springer, Cham, 2021. P. 143–150.

23. Bosilca G. et al. Parsec: Exploiting heterogeneity to enhance scalability //Computing in Science & Engineering. 2013. Т. 15. N 6. P. 36–45.

24. Малышкин В. Э. Технология фрагментированного программирования // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. 2012. № 46 (305).

25. Bichot C. E., Siarry P. (ed.). Graph partitioning. John Wiley & Sons, 2013.

26. Pellegrini F. Scotch and libScotch 5.1 user's guide. 2008.

27. Karypis G., Kumar V. METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. 1997.

28. Жданов А. И., Богданова Е. Ю. Об одной вычислительной реализации блочного метода Гаусса-Зейделя для нормальных систем уравнений // Вестник Самарского государственного технического университета. Серия: Физико-математические науки. 2016. Т. 20. № 4.

29. Davis T. A., Hu Y. The University of Florida sparse matrix collection //ACM Transactions on Mathematical Software (TOMS). 2011. Т. 38. N 1. P. 1–25.

30. Wang E. et al. Intel math kernel library // High-Performance Computing on the Intel§Xeon Phi^{TM}. Springer, Cham, 2014. P. 167–188.

31. [Electron. Res.]: https://www.jscc.ru/

32. Filippone S., Colajanni M. PSBLAS: A library for parallel linear algebra computation on sparse matrices // ACM Transactions on Mathematical Software (TOMS). 2000. Т. 26. N 4. P. 527–550.

Беляев Н. А. окончил бакалавриат Новосибирского государственного технического университета в 2015 году, окончил магистратуру Новосибирского государственного технического университета с отличием в 2017 году и в этом же году поступил в аспирантуру Института вычислительной математики и математической геофизики СО РАН в лабораторию синтеза параллельных программ. В процессе работы над кандидатским исследованием разработал и реализовал в виде компилятора языка LuNA системные алгоритмы конструирования высокоэффективных параллельных программ по описанию численного алгоритма. Окончил аспирантуру в 2021 году. Область научных интересов: автоматическое конструирование параллельных программ для гетерогенных параллельных компьютеров с распределенной памятью.

N. A. Belyaev received a bachelor's degree in applied mathematics and computer science from the Novosibirsk State Technical University in 2015. In 2017 N. A. Belyaev received a master's degree in applied mathematics and computer science from the Novosibirsk State Technical University. In the same year entered the graduate school of the Institute of Computational Mathematics and Mathematical Geophysics SB RAS (ICM&MG SB RAS) in the supercomputer software department. In the graduate school worked on specialized system algorithms of automated parallel program construction for the LuNA parallel programming system which is developed in the department. Research interests include parallel programming and automated parallel programs construction.

Дата поступления — 01.02.2022