УДК 681.3

А.А. Сиднев, В.П. Гергель

Нижегородский государственный университет им. Н.И. Лобачевского

ПРИМЕНЕНИЕ МАКРОМОДУЛЬНОГО ПОДХОДА К РАЗРАБОТКЕ ПРОГРАММ

Рассматривается макромодульный подход к разработке программ, который позволяет снизить сложность миграции на новые библиотеки и выбор оптимальной библиотеки. Приводятся краткое описание макромодульного подхода, методики его применения и основные полученные результаты. Накладные расходы на использование макромодульного подхода на тестовых вычислительных системах не превосходят 1 мс. Автоматический выбор наиболее эффективной реализации с использованием планировщика осуществляется более чем в 80 % случаев, а потери времени от ошибочного выбора не превосходят 6 %. Использование макромодульного подхода позволяет сократить время миграции на новую библиотеку более чем в 4 раза.

Ключевые слова: стандартизация, библиотеки, выбор оптимальной реализации, миграция, умножение матриц.

A.A. Sidnev, V.P. Gergel

Lobachevsky State University of Nizhni Novgorod

MACROMODULE APPROACH: APPLICATION TO PROGRAM DEVELOPMENT

Macromodular approach of development of programs which allows to reduce complexity of migration on new libraries and a choice of optimum library is considered. The short description of macromodular approach, a technique of its application and the main received results will come in dream. Overhead costs of use of macromodular approach on test computing systems do not exceed 1 ms. The automatic choice of the most efficient realization with use of the scheduler is carried out more than in 80% of cases, and losses of time from an inaccurate choice do not exceed 6 %. Use of macromodular approach allows to reduce migration time for new library more than by 4 times.

Keywords: standardization, library, selection of the optimal implementation, migration, matrix multiplication.

Введение. Для снижения сложности разработки программного обеспечения (ПО) общепринят модульный подход. Полученные модули могут быть использованы повторно, будучи оформленными в виде отдельных библиотек. В то же время постоянное развитие вычислительных систем [1], операционных систем и системного программного обеспечения приводит к широкому разнообразию вычислительных библиотек.

Один из нерешённых вопросов в модульном подходе к разработке программ – отсутствие стандартов на интерфейсы модулей. Разработчик библиотеки сам определяет удобные ему структуры хранения данных и интерфейсы функций, которые их обрабатывают. В результате каждая библиотека получается уникальной. Наличие большого количества библиотек приводит к необходимости выбора из них одной или нескольких наиболее подходящих. Поддержка нескольких библиотек усложняет структуру разрабатываемого проекта, а значит, трудозатраты на его разработку увеличиваются. Задача выбора библиотеки часто является многокритериальной, в связи с чем приходится идти на компромисс. Кроме того, в процессе разработки могут возникнуть задачи, которые нельзя решить, используя текущую библиотеку. При переходе к использованию новой библиотеки разработчику придётся столкнуться с необходимостью выполнить модификацию разработанных структур данных и функций под те, которые используются в библиотеке. Итак, основной недостаток классической модульной разработки программного обеспечения – это сильная зависимость от конкретной реализации библиотеки. Этот недостаток порождает три актуальные проблемы, которые тесно связаны и часто возникают одновременно:

– выбор наиболее подходящей библиотеки под текущие задачи проекта;

– поддержка нескольких библиотек;

– миграция на новую библиотеку.

Одно из лучших решений проблемы миграции – это разработка стандартного интерфейса для библиотек, решающих задачи одного класса. В результате все библиотеки реализуют одинаковый интерфейс, и задача перехода с одной библиотеки на другую решается просто (ярким примером является стандарт MPI [2]). У такого решения есть важный недостаток, который ограничивает его применимость, – разработка стандарта требует существенных усилий большой группы людей и занимает длительное время.

Сложность миграции приложения на новую библиотеку в первую очередь определяется качеством его проектирования и тем, была ли заложена при этом возможность миграции. Наиболее удачные решения, используемые при проектировании приложений, оформились в виде шаблонов проектирования [3]. С помощью предварительного анализа разрабатываемого ПО можно исключить потенциальные

проблемы совместимости. В работах [4, 5] рассматриваются инструмент и метод анализа бинарной совместимости разделяемых библиотек под Linux, а также набор правил, следование которым обеспечивает совместимость. В работе [6] приводится анализ мобильности приложений между различными версиями ОС Linux. Поддержка нескольких библиотек реализуется за счёт использования средств автоматизации сборки [7]. Широкое развитие получили средства автоматизации программного реинжиниринга. Набор инструментов DMS [8] позволяет выполнять автоматический анализ, трансформацию программ с одного языка программирования на другой и их синтез. Предметно-ориентированный язык RASCAL [9] позволяет выполнять анализ исходных кодов и автоматическую трансформацию. Универсальные языки TXL [10], Stratego [11] позволяют преобразовывать исходные коды программ за счёт манипуляции нотациями языков программирования (описываются грамматика текущего языка и правила модификации синтаксиса к требуемому). Когда приложение работает с конкретной библиотекой, можно использовать шаблонный метод трансформации программ [12], который заключается в формализации структуры и поведения исходной и целевой библиотек с помощью специального языка. На основе этих описаний выполняется трансформация программы под новую библиотеку.

Применение предлагаемых подходов требует специальных навыков, а по части рассмотренных направлений ведутся дальнейшие исследования. В данной работе рассматривается решение, объединяющее в себе часть рассмотренных подходов, что позволяет устранить их недостатки и снизить остроту перечисленных выше проблем. Данная работа является развитием [13].

Макромодульная разработка программ. Центральным элементом макромодульного подхода является наличие стандартного интерфейса для реализаций библиотек (рис. 1). Это обеспечивает как бинарную совместимость библиотек, так и единообразие программного кода.

Обеспечить реализации существующих библиотек, решающих задачи одного класса, единым интерфейсом (даже при наличии стандарта) – тяжелая задача. Для решения этой проблемы мы предлагаем разрабатывать программы-переходники (адаптеры) для каждой библиотеки, которые обеспечат «стыковку» стандартных и используемых в библиотеке интерфейсов.

Для подготовки вызова модуля программной библиотеки в соответствии со стандартным интерфейсом необходимо выполнить описание модели вычислений. Чтобы избежать зависимости описания от конкретного языка программирования, оно выполняется на специальном макроязыке.

Рис. 1. Основные положения макромодульного подхода

Для выбора наиболее эффективной реализации для текущей программно-аппаратной платформы среда исполнения макромодульных программ должна содержать планировщик, определяющий лучшие реализации модулей. Выбор может осуществляться статически во время сборки программы или динамически во время её выполнения. Последнее позволяет ориентироваться при разработке на более широкий круг программно-аппаратных платформ, так как для каждой из них выбирается лучшая из доступных реализаций.

При использовании макромодульного подхода можно выделить несколько этапов, характерных для классической разработки программ (рис. 2):

1) описание на макроязыке в приложении пользователя использующихся алгоритмов и структур данных;

2) обработка исходных кодов программы с помощью макропроцессора;

3) исполнение макромодульной программы, содержащей макрокоманды на макроязыке;

4) автоматический или ручной выбор наиболее эффективной реализации;

5) вызов адаптеров для выбранной реализации. Преобразование форматов хранения данных, если требуется.

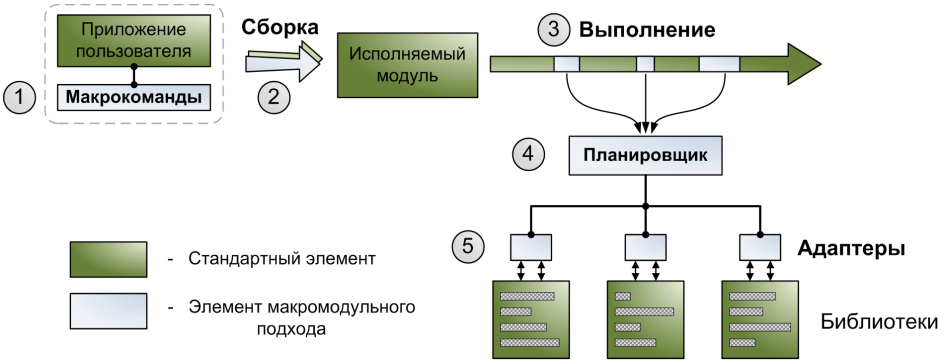

Рис. 2. Общая схема макромодульного подхода

Применение макромодульного подхода. Макромодульный подход содержит программные компоненты, которые обеспечивают быструю и простую замену используемых библиотек или поддержку нескольких библиотек. в то же время наличие этих компонент снижает производительность программы (результаты экспериментов будут приведены ниже), что не позволяет применять макромодульный подход для приложений ориентированных на максимальную производительность, таких как приложения реального времени. предлагаемый подход в первую очередь предназначен для разработки приложений, которые ориентированы на широкий круг поддерживаемых программно-аппаратных платформ, и для быстрой разработки прототипов.

Макромодульный подход предлагает готовые компоненты построения программного обеспечения: язык макроописаний, планировщик, средства сборки и выполнения приложений. Однако для разработки макропрограмм требуется предпринять ряд действий, специфичных для разных уровней пользователей (рис. 3):

1. Прикладной программист;
2. Системный администратор;
3. Инженер по сопровождению библиотек.

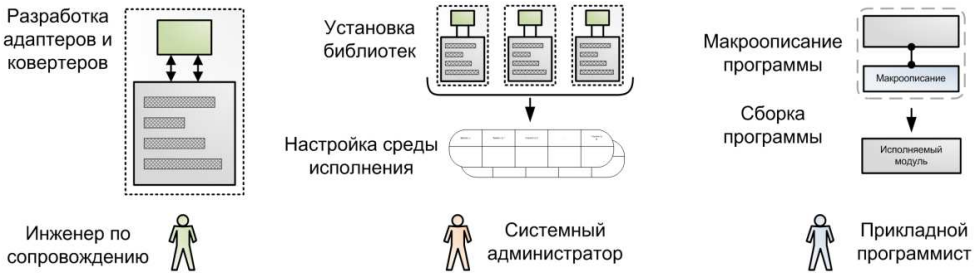

Рис. 3. Детализация действий пользователей макромодульного подхода

Инженер по сопровождению библиотеки выполняет реализацию адаптеров в соответствии с требованиями к их реализации, освобождая основных разработчиков библиотеки от данной работы. В том случае, если в библиотеке используются специфичные структуры данных, то для них потребуется реализация конвертеров. Разработанные адаптеры и конвертеры поставляются вместе с библиотекой. На практике эту роль может выполнять разработчик библиотеки.

Системный администратор выполняет установку библиотек и настройку среды исполнения макропрограмм на конкретной вычислительной системе.

Программист выполняет макроописание прикладной программы на макроязыке, сборку и запуск макропрограммы. При необходимости он может выбрать библиотеку и реализацию, которую необходимо использовать, иначе она определяется автоматически во время работы макропрограммы. Каждый независимый вычислительный участок программы описывается отдельно, формируя макрокоманду. Генерация макрокоманд может осуществляться автоматически специальной программой по параметрам задачи. Пример макрокоманды представлен на рис. 4.

Каждая макрокоманда применяется к блоку кода программы и задаётся в виде аннотации. Аннотация оформляется как комментарий языка программирования. Макрокоманда разделена на два блока. В первом блоке содержится описание вычислительной модели на макроязыке. Блок начинается со сточки комментария /** MMT begin и заканчивается */. Далее следует программный код, которому соответствует макрокоманда. В конце этого участка кода должен содержаться комментарий /** MMT end */, который является признаком конца макрокоманды.

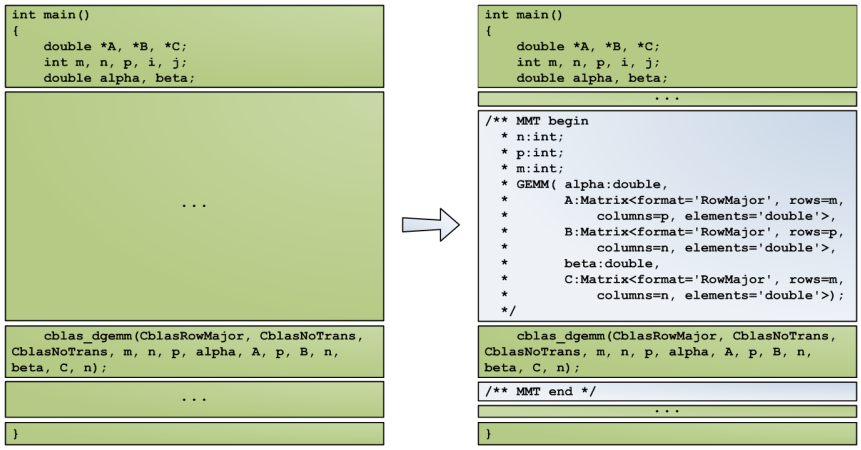

Рис. 4. Макрокоманда для умножения матриц (GEMM)

Планировщик. Планировщик выполняет оценку времени выполнения реализаций алгоритмов и выбор наиболее эффективной реализации среди множества доступных в системе. За счёт этого, разработчик освобождается от необходимости выбора лучшей реализации для заданных параметров задачи.

В работе [14] предложен метод оценки времени выполнения программ, который используется в реализации планировщика. Предложенный метод строит оценки без проведения экспериментов на вычислительной системе, используя результаты экспериментов, полученные на других вычислительных системах. Подобный подход позволяет не проводить длительное тестирование библиотек на конечной машине пользователя. Кроме того, предложенный метод способен выполнять экстраполяцию за пределами тестовой выборки и, как следствие, успешно применяться при наличии небольшого количества экспериментальных данных для задач малой размерности.

Результаты экспериментов на 84 вычислительных системах, которые были построены на процессорах разных поколений (от Pentium 4 до современных на базе ядра Haswell), разных производителей (Intel, AMD), разного назначения (мобильные, настольные, серверные) и, как следствие, существенно отличающихся производительностью, демонстрируют высокую эффективность работы планировщика. Оценка выполняется на основании 133 признаков вычислительных систем: 77 статических признаков описывают вычислительные способности процессора, 56 измеряемых признаков описывают пропускную спо-

собность подсистемы памяти. Эксперименты на 4 библиотеках с использованием случайного леса [15] показали, что средняя относительная ошибка предсказания не превосходит 8 % для матричного умножения, 3 % для сортировки, 11 % для решения СЛАУ, 23 % для быстрого преобразования Фурье. Предложенный метод оценки позволяет выбрать наиболее эффективную реализацию более чем в 80 % случаев, а потери времени от ошибочного выбора не превосходят 6 %.

Для оценки временных затрат на работу планировщика было разработано 4 приложения, которые выполняют умножение матриц:

1) реализация с использованием библиотеки MKL без использования макромодульного подхода. Накладные расходы отсутствуют;

2) реализация с использованием макромодульного подхода, в которой задана конкретная функция в адаптере MKL. Минимальный вклад планировщика во время работы программы, так как конкретная реализация выбрана разработчиком;

3) реализация с использованием макромодульного подхода, в которой задан адаптер MKL. Небольшой вклад планировщика во время работы программы, так как разработчиком задан адаптер, из которого выполняется выбор реализации;

4) реализация с использованием макромодульного подхода, в которой выполняется автоматический выбор реализации. Выполняется выбор среди всех доступных адаптеров, самый большой вклад планировщика во время работы программы.

На рис. 5 приведены результаты экспериментов на вычислительной системе с процессором Intel Core i7-3820 3.6 ГГц, 16 ГБ оперативной памяти и двумя графически картами NVIDIA GeForce GTX 680. При решении задач малой размерности планировщик вносит существенный вклад во время выполнения программы. Однако накладные расходы на использование макромодульного подхода не превосходят 1 мс. Это позволяет без существенной потери эффективности решать задачи средней и большой размерности. При этом эффективного выполнения задач малой размерности можно достигнуть за счёт программной оптимизации средств исполнения макромодульных программ и устранения зависимости от внешних сервисов. Отметим, что для матриц большой размерности планировщик в автоматическом режиме выбрал реализацию с использованием библиотеки cuBLAS, что позволило сократить время выполнения программы с 680 до 514 мс.

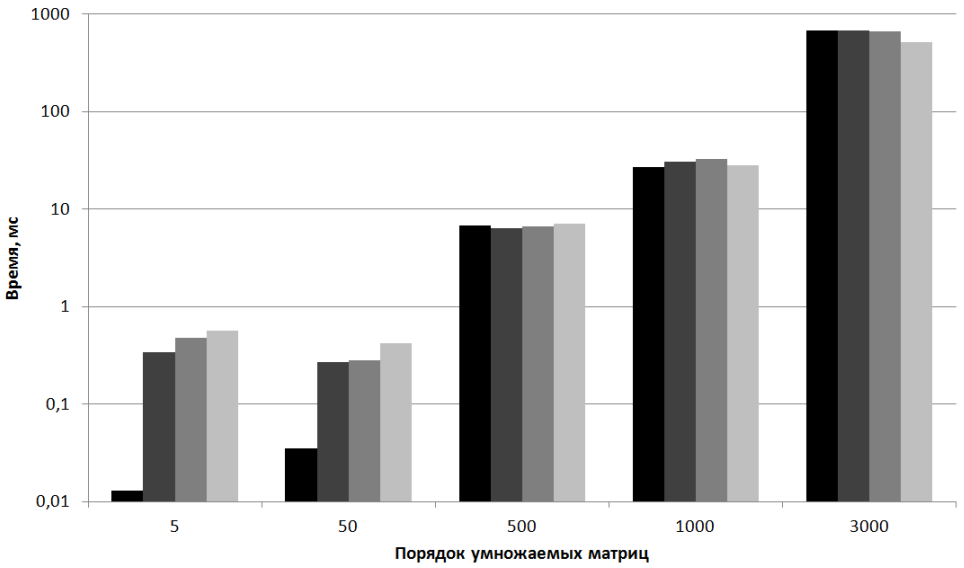

Рис. 5. Время умножения матриц: ■ – не используется макромодульный подход; ■ – задана конкретная функция в адаптере MKL; ■ – задан адаптер MKL; ■ – полностью автоматический выбор

Апробация макромодульного подхода. Для апробации макромодульного подхода были разработаны все необходимые компоненты:

– макропроцессор, выполняющий обработку программных кодов, которые содержат макрокоманды;

– модуль интеграции в среду разработки Microsoft Visual Studio 2012, который обеспечивает удобную сборку макропрограмм;

– средства исполнения макромодульных программ (планировщик, адаптеры и конвертеры).

Для оценки эффективности использования макромодульного подхода выполнено исследование трудозатрат при разработке и модификации ПО на примере задачи умножения матриц. Для исследования было привлечено несколько групп студентов факультета вычислительной математики и кибернетики ННГУ, начиная со 2-го курса и заканчивая 1-м курсом магистратуры. Всего в исследовании принимали участие 94 студента. Каждый студент выполнял разработку программы и последующий переход на заданную библиотеку с использованием, и без использования макромодульного подхода.

Каждый студент принимал участие в одном из 2 экспериментов, заключающихся в разработке 5 программ:

1) реализовать программу, выполняющую умножение матриц с заданным форматом хранения матриц (плотное по строкам);

2) реализовать программу, выполняющую умножение матриц с использование макромодульного подхода;

3) реализовать программу, выполняющую умножение матриц с использованием заданной библиотеки (Intel MKL);

4) реализовать программу, выполняющую умножение матриц с использованием макромодульного подхода для заданной библиотеки (Intel MKL);

5) модифицировать формат хранения матриц для заданной программы, выполняющей матричное умножение. В исходной программе выполнялось умножением в координатном формате. Требовалось перейти к формату CRS.

Участники эксперимента № 1 выполняли реализацию программ в том порядке, в котором они приведены выше. Участники эксперимента № 2 выполняли разработку программ в следующем порядке: 2, 1, 4, 3, 5. Такой подход к проведению экспериментов позволяет оценить влияние опыта разработки программы на её разработку другими методами.

Все участники эксперимента были условно разделены на две группы: сильные и слабые (таблица). К первой группе относятся те, которые обладают хорошими навыками разработки программ на C/C++, работают по специальности, знакомы с решаемой задачей. Ко второй группе – все остальные.

Результаты апробации макромодульного подхода

Задача разработки программы	Решение без использования макромодульного подхода, мин:с			Решение с использованием макромодульного подхода, мин:с		
	Слабые	В среднем	Сильные	Слабые	В среднем	Сильные
Разработка исходной версии	43:05	**30:56**	25:19	16:25	**16:15**	16:12
Переход на макромодульную реализацию	–	–	–	6:28	**5:23**	4:30
Переход на ручную реализацию	50:01	**27:26**	22:55	–	–	–
Переход на библиотеку MKL	20:18	**18:47**	18:11	4:05	**4:35**	4:47
Модификация формата хранения	–	–	–	–	**14:34**	14:34

Отметим, что без использования макромодульного подхода 18 участников не справились с разработкой исходной программы. С использованием макромодульного подхода эта задача была решена всеми. Как видно из таблицы, использование макромодульного подхода позволило сократить время на первоначальную разработку почти в 2 раза, а время миграции сократилось более чем в 4 раза. Отметим, что уровень подготовки студентов не оказывал существенного влияния на время решения задачи при использовании макромодульного подхода.

Для апробации макромодульного подхода разработаны примеры с использованием функций BLAS и FFTW3. Реализованы адаптеры для библиотек MKL, OpenBLAS, cuBLAS и FFTW. Полученные примеры могут использовать требуемую библиотеку без модификаций исходного кода. Разработан унифицированный интерфейс для переупорядочивателей при решении СЛАУ. Выполнена адаптация под этот интерфейс существующих реализаций с использованием макромодульного подхода: METIS [16], O-MATRUZ [17], MORSY [18]. Панируется использовать предложенный подход при разработке программного обеспечения для решения задач глобальной оптимизации [19].

Заключение. На данный момент во многих предметных областях существует множество библиотек с реализацией различных алгоритмов. Отсутствие стандартов на интерфейсы этих библиотек порождает ряд проблем, связанных с поддержкой нескольких библиотек, миграцией на новые библиотеки и выбором наиболее подходящей под задачи проекта. Предложенный в работе макромодульный подход разработки программ позволяет снизить остроту перечисленных проблем и сократить трудозатраты на разработку программного обеспечения.

Выполнена реализация всех программных компонент, необходимых для сборки и исполнения макромодульных программ, и апробация предложенного подхода. Накладные расходы на использование макромодульного подхода на тестовых вычислительных системах не превосходят 1 мс. Автоматический выбор наиболее эффективной реализации с использованием планировщика осуществляется более чем в 80 % случаев, а потери времени от ошибочного выбора не превосходят 6 %. Использование макромодульного подхода позволило сократить время миграции на новую библиотеку более чем в 4 раза на тестовой задаче.

Работа поддержана грантом (соглашение от 27 августа 2013 г. № 02.B.49.21.0003 между МОН РФ и ННГУ).

Библиографический список

1. TOP 500. – URL: http://www.top500.org/lists/ (дата обращения: 30.09.2014).

2. MPI Forum. – URL: http://www.mpi-forum.org/ (дата обращения: 30.09.2014).

3. Design Patterns: Elements of Reusable Object-Oriented Software / E. Gamma, R. Helm, R. Johnson, J. Vlissides. Pearson Education. – 1994.

4. Shved P., Silakov D. Binary Compatibility of Shared Libraries Implemented in C++ on GNU/Linux Systems. SYRCoSE, 2009. – URL: http://syrcose.ispras.ru/2009/files/02_paper.pdf (дата обращения: 30.09.2014).

5. Ponomarenko A., Rubanov V., Khoroshilov A. A system for backward binary compatibility analysis of shared libraries in Linux. Proc. of Software Engineering Conference in Russia (CEE-SECR). – 2009. – P. 25–31.

6. Rubanov V. Automatic Analysis of Applications for Portability Across Linux Distributions. Proc. of the Third International Workshop on Foundations and Techniques for Open Source Software Certification. – 2009. – Vol. 20. – P. 1–9.

7. Taylor I.L. The GNU configure and build system. – URL: http://airs.com/ian/configure/ (дата обращения: 30.09.2014).

8. DMS Toolkit. – URL: http://www.semdesigns.com/products/DMS/DMSToolkit.html (дата обращения: 30.09.2014).

9. Klint P., Storm T., Vinju J. RASCAL: A Domain Specific Language for Source Code Analysis and Manipulation. Proc. of Ninth IEEE International Working Conference on Source Code Analysis and Manipulation. – 2009. – P. 168–177.

10. Cordy J. R. The TXL source transformation language. Science of Computer Programming. – 2006. – N 61 (3). – P. 190–210.

11. Bravenboer M., Dam A., Olmos K. Eelco V. Program Transformation with Scoped Dynamic Rewrite Rules. Technical Report UU-CS-2005-005, department of Information and Computing Sciences. – 2005.

12. Ицыксон В., Зозуля А. Автоматизированная трансформация программ при миграции на новые библиотеки. Программная инженерия. – 2012. – № 6. – С. 8–14.

13. Гергель В.П., Сиднев А.А. Методы и программные средства макромодульной разработки программ // Вестник Нижегород. ун-та им. Н.И. Лобачевского. – 2012. – № 5(2). – С. 294–300.

14. Sidnev A.A., Gergel V.P. Automatic selection of the fastest algorithm implementations // Numerical Methods and Programming. – 2014. – Vol. 15. – P. 579–592.

15. RandomForest: Breiman and Cutler's random forests for classification and regression. – URL: http://cran.r-project.org/web/packages/randomForest/index.html (дата обращения: 6.04.2014).

16. Karipis G. METIS. A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices. Version 5.0. Technical report. – University of Minnecota, Department of Computer Science and Engeneering. – 2011.

17. Старостин Н.В., Филимонов А.В. Разработка и реализация параллельного многоуровневого алгоритма равномерного разбиения нераспределенного графа // Сборник трудов HPC. – 2013. – С. 243–248.

18. Pirova A., Meyerov I. MORSy – a new tool for sparse matrix reordering. An International Conference on Engineering and Applied Sciences Optimization. – Kos Island, Greece, 4–6 June 2014. 1952–1964.

19. Gergel V.P., Sergeyev Ya.D. Sequential and parallel algorithms for global minimizing functions with lipschitzian derivatives. Computers & Mathematics with Applications. – 1999. – Т. 37. – № 4–5. – P. 163–179.

References

1. TOP 500. Available at: http://www.top500.org/lists/ (accessed: 30.09.2014).

2. MPI Forum. Available at: http://www.mpi-forum.org/ (accessed: 30.09.2014).

3. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Pearson Education, 1994.

4. Shved P., Silakov D. Binary Compatibility of Shared Libraries Implemented in C++ on GNU/Linux Systems. SYRCoSE, 2009. Available at: http://syrcose.ispras.ru/2009/files/02_paper.pdf (accessed: 30.09.2014).

5. Ponomarenko A., Rubanov V., Khoroshilov A. A system for backward binary compatibility analysis of shared libraries in Linux. *Proc. of Software Engineering Conference in Russia* (CEE-SECR), 2009, pp. 25-31.

6. Rubanov V. Automatic Analysis of Applications for Portability Across Linux Distributions. *Proc. of the Third International Workshop on Foundations and Techniques for Open Source Software Certification*, 2009, vol. 20, pp. 1-9.

7. Taylor I.L. The GNU configure and build system. Available at: http://airs.com/ian/configure/ (accessed: 30.09.2014).

8. DMS Toolkit. Available at: http://www.semdesigns.com/products/DMS/ DMSToolkit.html (accessed: 30.09.2014).

9. Klint P., Storm T., Vinju J. RASCAL: A Domain Specific Language for Source Code Analysis and Manipulation. *Proc. of Ninth IEEE International Working Conference on Source Code Analysis and Manipulation*, 2009, pp. 168-177.

10. Cordy J. R. The TXL source transformation language. *Science of Computer Programming*, 2006, no. 61(3), pp. 190-210.

11. Bravenboer M., Dam A., Olmos K. Eelco V. Program Transformation with Scoped Dynamic Rewrite Rules. Technical Report UU-CS-2005-005, department of Information and Computing Sciences, 2005.

12. Itsykson V., Zozulia A. Avtomatizirovannaia transformatsiia programm pri migratsii na novye biblioteki [Automated transformation of programs in case of migration on new libraries]. *Programmnaia inzheneriia*, 2012, no. 6, pp. 8-14.

13. Gergel' V.P., Sidnev A.A. Metody i programmnye sredstva makromodul'noi razrabotki programm [Methods and software of macromodular development of programs]. *Vestnik Nizhegorodskogo universiteta imeni N.I. Lobachevskogo*, 2012, no. 5(2), pp. 294-300.

14. Sidnev A.A., Gergel V.P. Automatic selection of the fastest algorithm implementations. *Numerical Methods and Programming*, 2014, vol. 15, pp. 579-592.

15. randomForest: Breiman and Cutler's random forests for classification and regression. Available at: http://cran.r-project.org/web/packages/randomForest/index.html (accessed: 6.04.2014).

16. Karipis G. METIS. A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices. Version 5.0. Technical report, University of Minnecota, Department of Computer Science and Engeneering, 2011.

17. Starostin N.V., Filimonov A.V. Razrabotka i realizatsiia parallel'nogo mnogourovnevogo algoritma ravnomernogo razbieniia neraspredelennogo grafa [Development and implementation of parallel multi-level algorithm of uniform partition of an unallotted graph]. *Sbornik trudov High Performance Centre*, 2013, pp. 243-248.

18. Pirova A., Meyerov I. MORSy – a new tool for sparse matrix reordering. *An International Conference on Engineering and Applied Sciences Optimization*. Kos Island, Greece, 4-6 June 2014, pp. 1952-1964.

19. Gergel V.P., Sergeyev Ya.D. Sequential and parallel algorithms for global minimizing functions with lipschitzian derivatives. *Computers & Mathematics with Applications,* 1999, vol. 37, no. 4-5, pp. 163-179.

Сведения об авторах

Сиднев Алексей Александрович (Нижний Новгород, Россия) – ассистент факультета вычислительной математики и кибернетики Нижегородского государственного университета им. Н.И. Лобачевского (603950, г. Нижний Новгород, пр. Гагарина, 23, e-mail: alexey.sidnev@gmail.com).

Гергель Виктор Павлович (Нижний Новгород, Россия) – доктор технических наук, профессор факультета вычислительной математики и кибернетики Нижегородского государственного университета им. Н.И. Лобачевского (603950, г. Нижний Новгород, пр. Гагарина, 23, e-mail: gergel@unn.ru).

About the authors

Sidnev Alexey Aleksandrovich (Nizhny Novgorod, Russian Federation) the assistant Faculty of Computational Mathematics and Cybernetics to N.I. Lobachevsky State University of Nizhni Novgorod (603950, Nizhny Novgorod, pr. Gagarina, 23, e-mail: alexey.sidnev@gmail.com).

Gergel Viktor Pavlovich (Nizhny Novgorod, Russian Federation) Doctor of Technical Sciences, Professor Faculty of Computational Mathematics and Cybernetics to N.I. Lobachevsky State University of Nizhni Novgorod (603950, Nizhny Novgorod, pr. Gagarina, 23, e-mail: gergel@unn.ru).

Получено 12.12.2014