УДК 004.942

С.В. Рогачев[1]

ОПТИМИЗАЦИЯ ПРОИЗВОДИТЕЛЬНОСТИ ПРОГРАММНОГО КОМПЛЕКСА ДЛЯ РАСЧЕТА СПЕЦИАЛЬНЫХ ФУНКЦИЙ ВОЛНОВЫХ КАТАСТРОФ

В статье рассматривается вопрос развития комплекса программ для численного моделирования СВК, в частности – оптимизация производительности различных компонент системы: вычислительного ядра и модуля построения контурных графиков. Рассмотрены недавние изменения, связанные с сохранением значений СВК в различных точках для последующего использования при решении задач со схожими входными параметрами, а также новая возможность построения распределенной вычислительной сети.

Ключевые слова: *компьютерное моделирование, СВК, теория катастроф, оптимизация программного обеспечения.*

S.V. Rogachev

PERFORMANCE OPTIMIZATION OF THE SOFTWARE FOR COMPUTING THE SPECIAL FUNCTIONS OF WAVE CATASTROPHES

In the article, the author enlightens a question of development of the software for numeric modeling of SWCs, in particular ways of performance optimization used for various components of the system: the computing core and the contour-plotter component. Recent changes in the source code are overviewed, for example, the things related to caching SWC values in different points after calculation for future use in similar tasks, or a new possibility to setup the software to serve requests as a kind of distributed computing network.

Keywords: *computer modeling, SWC, theory of catastrophes, software optimization.*

1. Введение в предметную область

Текст настоящей статьи посвящен вопросу оптимизации производительности комплекса программ для численного моделирования специальных функций волновых катастроф (СВК). СВК – спецфункции, используемые в равномерной асимптотической теории основных, краевых и угловых волновых катастроф при построении равномерных асимптотических решений задач радиофизики. При наличии некоторой особенности фокусировки Σ, равномерная асимптотика поля в окрестности особенности этого типа имеет вид [1]:

$$U\left(\Lambda,\vec{\alpha}\right)\cong\exp\left(i\theta\right)\left[l_1 I^{\Sigma}\left(\vec{S}\right)+\sum_{j=2}^{N} l_j\frac{\partial I^{\Sigma}}{\partial S_{j-1}}\right], \quad (1)$$

где l_j – коэффициенты асимптотического разложения, имеющие зависимость от большого параметра Λ (по обратным степеням), I^{Σ} – СВК, соответствующая особенности Σ, также можно заметить первые частные производные СВК по S_{j-1}.

[1] Аспирант АНО ВО «Российский новый университет».

Здесь

$$\theta=\Lambda\tilde{\theta}\left(\vec{\alpha}\right),\ \vec{S}=\left(\vec{\lambda},\vec{\alpha}\right). \quad (2)$$

Непосредственно СВК имеют вид:

$$I^{\Sigma}\left(\vec{\lambda},\vec{\alpha}\right)=\int_{-\infty}^{+\infty}\exp\left[iF_{\Sigma}\left(\vec{x},\vec{\lambda},\vec{\alpha}\right)\right]dx. \quad (3)$$

В интеграле выше F_{Σ} – универсальная деформация, устойчивое функциональное семейство, соответствующее особенности дифференцируемого отображения.

Комплекс программ, упоминаемый в статье, позволяет вычислять СВК в различных точках $\vec{S}$, то есть при разных значениях $\vec{\lambda}$ и $\vec{\alpha}$.

2. Введение в программный комплекс и его построение

Описываемый комплекс программ многократно упоминался в публикациях, в которых разбирались математические методы, используемые в нем для вычисления спецфункций, алгоритмические и программные решения (см. [2–7]).

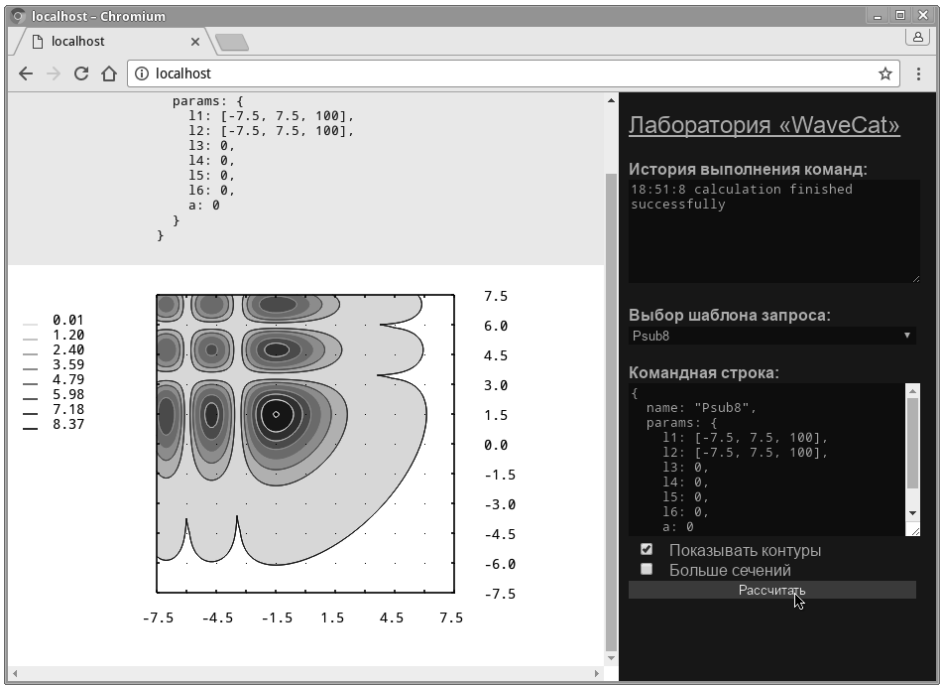

Рис. 1. Интерфейс системы в браузере Chromium с результатами вычисления СВК особенности P_8

Следующая диаграмма компонентов UML (рис. 2) представляет основные компоненты системы и пути их взаимодействия.

Система представлена тремя основными группами компонентов: клиентом, сервером и языковым инструментарием. Клиентские компоненты включают в себя интерфейс для диалогового взаимодействия с пользователем и ядро визуализации, ответственное за графическое представление результатов моделирования СВК.

Серверные компоненты – основа системы, это HTTP-сервер, осуществляющий взаимодействие с клиентом, и вычислительное ядро, производящее необходимые вычисления, используя метод ОДУ [8]. Производительность этого компонента определяет производительность всего комплекса программ в целом.

К языковому инструментарию относится транслятор предметно ориентированного языка (DSL), на котором возможно записывать про-

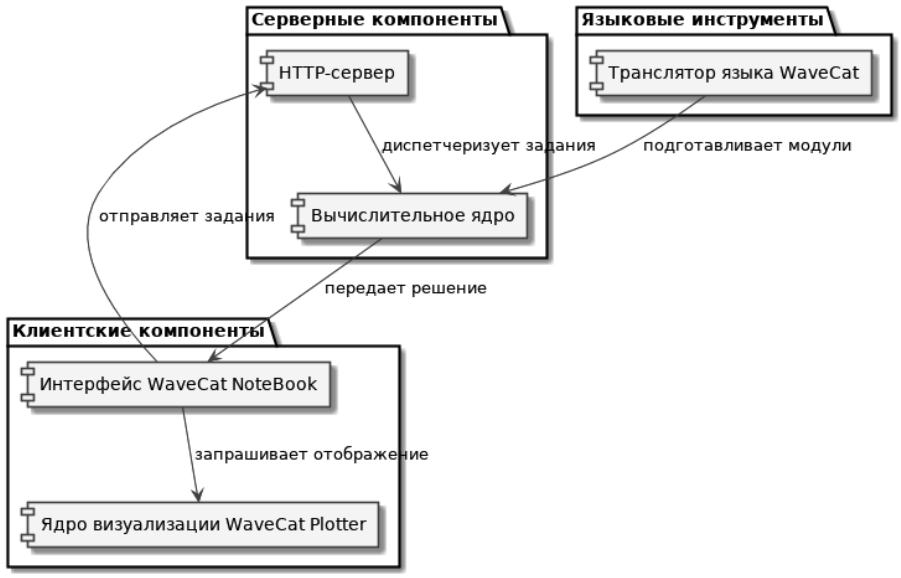

Рис. 2. Компонентная UML диаграмма, связывающая составные части комплекса программ

граммы моделирования СВК отвлеченно от особенностей реализации вычислительного ядра. Язык в определенном смысле имеет схожие черты с языками школы Никлауса Вирта (Паскаль, Оберон, Модула).

Настоящая статья преимущественно сфокусирована на задаче оптимизации производительности комплекса программ за счет применения параллелизма, распределенных вычислений, динамического программирования (кеширования результатов), а также низкоуровневых аппаратно-специфических оптимизаций.

3. Оптимизация вычислительного ядра

3.1. Параллелизм

Как стало ясно из параграфа 1, при решении задач радиофизики методами равномерной асимптотической теории волновых катастроф требуется вычислять специальную функцию волновых катастроф (СВК) и ее первые производные в отдельных точках (при различных значениях параметров $\vec{\lambda}$ и функционального модуля $\vec{\alpha}$). Очевидно, что вычисления в различных точках не зависят друг от друга и могут быть выполнены на различных вычислительных устройствах.

Возможны два типа декомпозиции в зависимости от типа решаемой задачи. В демонстрационных целях СВК и ее производные могут вычисляться при фиксированном шаге изменения параметров, что позволяет произвести анализ эталонной структуры поля, описываемой данной спецфункцией. В таком случае производится декомпозиция по параметру с наибольшим числом шагов.

В реальном приложении параметры СВК связываются с параметрами исходной задачи, что является отдельной проблемой и выходит за рамки данной работы. В таком случае требуется производить вычисления в несвязанных точках. Это приводит к необходимости хранения информации о каждой точке и группирования этих данных в контейнерах, ассоциированных с различными вычислительными устройствами. Такой подход поднимает вопрос борьбы с ложным разделением кеш-линий ядер процессора (false sharing), который рассматривается в пункте 3.3.

Идея декомпозиции задачи для параллельного решения уже рассматривалась в публикациях автора, данная статья представляет результаты на примере расчета СВК нескольких особенностей.

Таблица 1 представляет эффект параллельного решения на компьютере с процессором AMD Thurion II P520 (два ядра, 4788 bogomips).

Таблица 1

Наблюдаемый эффект параллельного решения (1,8 раз)

Измерение	Последовательное вычисление P_8 (мс)	Параллельное вычисление P_8 (мс)	Последовательное вычисление E_6 (мс)	Параллельное вычисление E_6 (мс)
1	13963	7603	3969	2065
2	13951	7622	3774	2079
3	14000	7645	3766	2062
4	14009	7612	3767	2048
5	13920	7867	3775	2078
Среднее	13969	7670	3810	2066
Сред. откл.	28,72	78,88	63,52	9,68

Аналогичная таблица представлена для другого оборудования, AMD Phenom(tm) II X4 (четыре ядра, 5619 bogomips).

Таблица 2

Наблюдаемый эффект параллельного решения (3,3 раза)

Измерение	Последовательное вычисление P_8 (мс)	Параллельное вычисление P_8 (мс)	Последовательное вычисление E_6 (мс)	Параллельное вычисление E_6 (мс)
1	3393	1013	926	276
2	3389	1026	925	281
3	3390	1019	925	278
4	3389	1029	924	293
5	3408	1017	951	282
Среднее	3394	1021	930	282
Сред. откл.	5,68	5,36	8,32	4,4

3.2. Динамическое программирование

Результаты профилирования вычислительного ядра подтверждают факт, что наибольшее время тратится в функции численного решения системы обыкновенных дифференциальных уравнений, соответствующей интегралу СВК (в зависимости от особенности может применяться как метод Рунге – Кутты, так и метод Кутты – Мерсона). В определенном смысле дальнейшая значительная оптимизация этой части сложна, поэтому требуется иной подход.

Вычисление СВК имеет смысл только на определенном множестве точек, поскольку в точках, лежащих за пределами этого множества, спецфункция расходится. Следовательно, при многократном вычислении СВК в различных приложениях становится возможным повторно использовать уже полученные значения СВК (вероятно, они уже вычислялись), не тратя времени на «дорогостоящий» численный метод. Поскольку решение задачи в данном случае эквивалентно решению всех подзадач – нахождению СВК во всех требуемых точках, то сохранение рассчитанных значений СВК позволяет в дальнейшем сократить число решаемых подзадач. Этот подход аналогичен тому, что в теории алгоритмов называется динамическим программированием.

Для того чтобы эффективно сохранять результаты вычислений, а также быстро находить соответствующее значение СВК, необходимо выполнение двух условий.

1. Вычислительное ядро не должно завершать свое исполнение после выполнения задания, чтобы полученные решения подзадач хранились в оперативной памяти.

2. Требуется контейнер с эффективным добавлением и поиском элементов, где ключом являлась бы точка, а данными – значение спецфункции в этой точке.

Наиболее подходящим контейнером в данном случае является красно-черное дерево (см., например, [9]), относящееся к самобалансирующимся двоичным деревьям поиска. Среднее, как и наихудшее, время поиска операций с этим деревом асимптотически оценивается как $O(\log N)$, где N, применительно к рассматриваемой задаче, – число точек, информация о которых уже добавлена в контейнер.

Тем не менее, требуется ряд модификаций. Необходимо различать значения разных спецфункций, таким образом, требуется использовать лес двоичных деревьев поиска, где корень каждого дерева соответствует отдельной СВК.

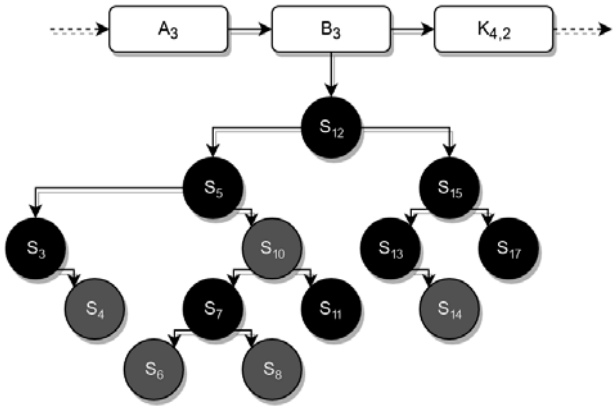

Рис. 3. Пример контейнера в работе: каждой особенности соответствует свое красно-черное дерево, элементы хранятся в соответствии с требованиями к ключам бинарного дерева поиска. Обязательные черные листовые узлы опущены

Также необходимо обеспечить хранение значений СВК по составному ключу, поскольку отдельные точки по своей сути являются векторами. То есть, для точек необходимо ввести функцию $\mathrm{Ord}(S_i, S_j)$, осуществляющую их сравнение при операции поиска.

```
Ord(S_i, S_j) =
для каждой пары элементов e_i и e_j векторов S_i и S_j:
        если e_i ! = e_j:
                если e_i < e:
                        возвратить –1
                иначе
                        возвратить 1
возвратить 0
```

Псевдокод 1. Пример алгоритма для сравнения составных ключей при поиске значения СВК в дереве

Размерности векторов S_i и S_j одинаковы, поскольку относятся к одной и той же спецфункции.

Важно заметить, что при реализации на компьютере нельзя производить прямое сравнение значений вещественных и комплексных чисел в силу особенностей их представления. Таким образом, требуется производить сравнение с точностью до некоторой постоянной ε.

```
Eq(z_i, z_j) =
        если |z_i – z_j| < Epsylon:
                возвратить 1
        иначе
                возвратить 0
```

Псевдокод 2. Пример способа сравнения «машинных» комплексных и вещественных чисел

Следующая таблица отражает результаты замеров производительности после применения описанного подхода. Выполняется расчет значений СВК $K_{4,2}$ на заданном интервале, время вычислений измеряется. Далее задача решается на больших интервалах (в большем числе точек), при этом, несмотря на двухкратный и четырехкратный рост размера задачи, время, требуемое для решения, – уменьшается.

Таблица 3

Эффект от применения метода динамического программирования (кеширования результатов)

Измерение	Наблюдаемое время (мс)
Вычисление СВК $K_{4,2}$ на заданном интервале	8467
Повторное вычисление на том же интервале	31
Вычисление на интервале вдвое большем, чем изначальный по λ_1	6576
Вычисление на интервале вдвое большем, чем изначальный по λ_1 и λ_2	4298

3.3. Низкоуровневая оптимизация

Как было замечено в параграфе 3.1., при параллельном расчете СВК необходимо правильно организовать в памяти структуры данных, хранящие информацию, необходимую для вычисления спецфункции в каждой точке, а также сам результат вычислений. При этом требуется задумываться о негативных эффектах кеш-памяти, проявляющихся при параллельном программировании. Проблема ошибочного разделения кеш-линии (cache false sharing) может приводить к катастрофическим падениям производительности [10] (2-3 раза). Иллюстрация ниже показывает такую возможность.

Предположим, что данные, связанные с двумя различными точками, находятся в пределах одной кеш-линии, и СВК в каждой из этих точек рассчитывается разными ядрами процессора. Модификация значений в этих данных каждым из процессоров приводит к инвалидации кеш-линии, поскольку копии в локальных по отношению к ядрам в кешах первого и второго уровней становятся неверными, и требуется перечитывание линии из более медленной памяти следующей иерархии.

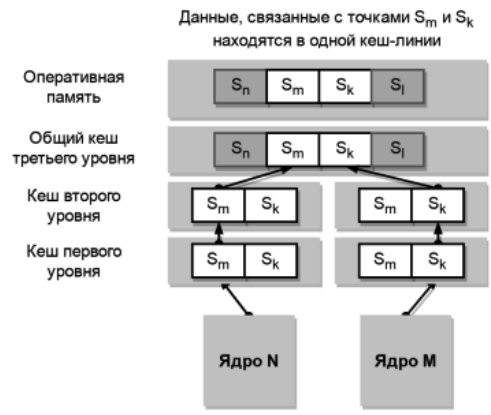

Рис. 4. Иллюстрация ошибочного разделения кеш-линии; данные, связанные с двумя точками, обрабатываемыми разными ядрами процессора, конкурентно модифицируются, что приводит к частой инвалидации кеш-линии

Для уменьшения описанного эффекта необходимо гарантировать, что данные, модифицируемые различными потоками исполнения, находятся на разных кеш-линиях. Это достигается путем выделения памяти для разных ядер процессора большими блоками с обеспечением выравнивания по границе кеш-линии. Возможны и другие подходы.

3.4. Распределенные вычисления

Как было замечено в параграфе 2, вычислительное ядро комплекса программ физически отделено от интерфейса пользователя и общается с ним опосредованно через HTTP-сервер, используя быстрый протокол SCGI (Simple Common Gateway Interface). Подобный подход позволяет преобразовать систему в распределенную вычислительную сеть. Это достигается запуском нескольких копий вычислительного ядра на разных вычислительных машинах в локальной сети или сети Интернет и соответствующей настройкой HTTP-сервера.

Автор использует HTTP-сервер Nginx, имеющий поддержку SCGI-протокола. Данный сервер позволяет настроить его в режим reverse proxy, осуществляющий балансировку нагрузки между несколькими SCGI-серверами. Таким образом, нагрузка на копии вычислительного ядра, запущенные на разных компьютерах, распределяется равномерно, и значительно увеличивается мощность (пропускная способность) такого «кластера».

4. Оптимизация клиентских компонент

4.1. Изображение контурных графиков

Для отображения результатов моделирования комплекс программ использует алгоритм на основе метода марширующих квадратов (marching squares в англоязычной литературе). Алгоритм представляет собой метод построения изолиний на двумерном скалярном поле, он достаточно распространен, а его корректность в более общем случае была доказана с использованием систем автоматизированного доказательства теорем (Coq) [11]. Кратко изложим суть алгоритма для случая контуров без заполнения, поскольку его описание представляется наименее громоздким.

Для применения алгоритма строится регулярная сетка, в узлах которой известно значение поля, в данном случае – значение СВК в соответствующей точке. В качестве сетки комплекс программ использует матрицу, содержащую решение задачи, полученную от вычислительного ядра. Ячейки этой матрицы содержат значения СВК, полученные при изменении параметров с некоторым фиксированным шагом.

Для каждого уровня h (секущей плоскости) ячейки сетки классифицируются путем сравнения значения поля в вершинах и значения уровня. Возможен ряд случаев: значение поля выше/ниже значения уровня во всех вершинах, значение поля выше/ниже значения уровня в одной вершине, значение поля выше/ниже уровня в соседних вершинах, значение поля выше/ниже уровня в противоположных вершинах. Эти четыре случая возможны в четырех ориентациях, то есть существует 16 классов ячеек. В зависимости от класса ячейки она изображается особым образом – ориентированным отрезком или парой отрезков, соединяющих точки на ребрах ячейки.

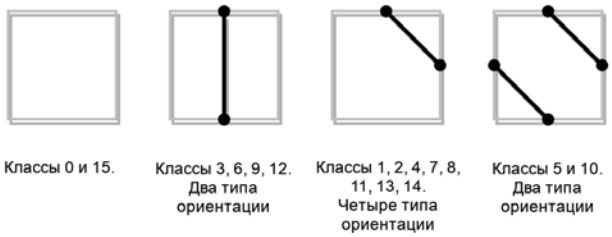

Рис. 5. Классы ячеек в алгоритме «марширующие квадраты» и соответствующие им образы на графике

Алгоритм редко используется в чистом виде, поскольку изолинии получаются ломаными. Для улучшения читаемости графика применяется линейная интерполяция, производимая для каждой точки на ребре сетки.

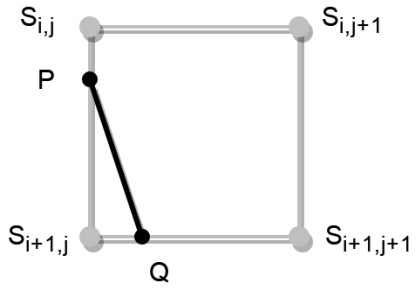

Рис. 6. Различные положения точек на ребрах ячейки при линейной интерполяции

Отношение, в котором точка Q делит отрезок $[S_{i+1,j}, S_{i+1,j+1}]$, определяется дробью:

$$\frac{U\left(S_{i+1,j}\right)-h}{h-U\left(S_{i+1,j+1}\right)}. \qquad (4)$$

Рассуждения для точки P аналогичны.

Предшествующая наивная реализация алгоритма имеет недостаток, поскольку производит интерполяцию для каждой ячейки сетки независимо. Сам алгоритм достаточно быстр, и оптимизации обычно не требуются, но поскольку в комплексе программ используется реализация на интерпретируемом языке JavaScript, решение оказывается достаточно медленным в случае большого количества точек – происходит заметная задержка при изображении графика.

Легко заметить, что наивно реализованный алгоритм выполняет лишнюю работу, поскольку производит интерполяцию для каждого ребра два раза – одинаковые операции производятся для каждой ячейки сетки. При этом для смежных ячеек отношение, полученное при интерполяции для их общего ребра, одинаково. Таким образом, достаточно сохранять значения, рассчитанные для правого ребра предыдущей левой ячейки, а также нижних ребер ячеек предшествующей линии сетки.

Заключение

Применение комплекса программ для одновременного обслуживания нескольких пользователей требует значительной работы в области снижения времени отклика системы, повышения ее надежности и безопасности. В настоящей статье были проиллюстрированы подходы, примененные в системе численного моделирования СВК с целью ускорения вычислений и уменьшения времени отклика. Эти изменения позволяют обрабатывать большее количество заданий от большего числа пользователей и приближают ее

к возможности одновременного использования в локальной сети университета или иной организации.

Например, сохранение результатов вычислений, описанное в параграфе 3.2., позволяет моментально получать решения для типовых заданий, а также значительно (в разы) снижать время расчета СВК на «нестандартных» диапазонах значений параметров, при условии, что часть диапазона когда-либо уже была вычислена.

Возможность запуска вычислительного ядра в качестве SCGI-приложения позволила предоставлять ресурс распределенной вычислительной сети, создавая иллюзию одного вычислительного сервера. Возможны конфигурации с произвольным количеством машин, выполняющих вычислительное ядро, в таком случае происходит балансировка нагрузки, и задания прозрачно перенаправляются к разным звеньям этой сети. Рассмотрены и иные виды оптимизации производительности, значительно улучшающие отзывчивость системы.

С небольшой задержкой последние изменения в системе появляются в публичном GIT-депозитарии исходного кода, доступном по ссылке: https://github.com/richiefreedom/wavecat. Код всего программного инструментария открыт под публичной лицензией GPL v3 и может быть легко собран для личного пользования.

Литература

1. Крюковский А.С. Равномерная асимптотическая теория краевых и угловых волновых катастроф. – М. : РосНОУ, 2013. – 386 с.

2. Крюковский А.С., Рогачев С.В Система расчета и визуализации специальных функций волновых катастроф // Электромагнитные волны и электронные системы. – 2013. – Т. 18. – № 8. – С. 10–17.

3. Крюковский А.С., Рогачев С.В. Разработка специализированной системы расчета специальных функций волновых катастроф // III Всероссийские Армандовские чтения [Электронный ресурс]: Сверхширокополосные сигналы в радиолокации, связи и акустике // Материалы IV Всероссийской научной конференции (Муром, 25–27 июня 2013 г.). – Муром : Изд.- полиграфический центр МИ ВлГУ, 2013. – 271 с. – С. 129–144.

4. Крюковский А.С., Рогачев С.В. Специализированная система расчета и визуализации специальных функций волновых катастроф // Цивилизация знаний: проблемы и смыслы образования : Труды XIV Международной научной конференции, Москва, 26–27 апреля 2013 г. – М. : РосНОУ, 2013. – Часть 2. – С. 117–174.

5. Крюковский А.С., Рогачев С.В. Архитектура программного комплекса расчета специальных функций волновых катастроф // Вестник Российского нового университета. – 2014. – Выпуск 4. Управление, вычислительная техника и информатика. – С. 13–20.

6. Рогачев С.В. Проектирование предметно ориентированного языка для расчета специальных функций волновых катастроф // Вестник Российского нового университета. – 2013. – Выпуск 4. Управление, вычислительная техника и информатика. – С. 47–52.

7. Kryukovsky, A.S., Rogachev, S.V., Lukin, D.S. Special Software for Computing the Special Functions of Wave Catastrophes // Revista de Matematica: Teoria y Aplicaciones / San Pedro Montes de Oca, San Jose, Costa Rica : Universidad de Costa Rica, 2015. – Vol. 22. – Num. 1. – P. 21–30.

8. Крюковский А.С. Метод обыкновенных дифференциальных уравнений для расчета специальных функций волновых катастроф (СВК) // Дифракция и распространение электромагнитных волн : межведомственный сборник научных трудов. – М. : Московский физико-технический институт, 1992. – С. 29–48.

9. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ. – 2-е изд. – М. : Вильямс, 2005. – 1296 с.

10. Torellas, J., Lam, M., Hennessy, J. False Sharing and Spatial Locality in Multiprocessor Caches. – IEEE : Transactions on Computers, 1994. – Vol. 43. – Num 6. – P. 651–663.

11. Chernikov, A., Xu, J. Proof of Correctness of a Marching Cubes Algorithm Carried out with Coq, Computational Geometry Theory and Applications. – Department of Computer Science, Old Dominion University, 2015.