УДК.537.87:621.396.67

А.С. Крюковский[1]
С.В. Рогачев[2]

АРХИТЕКТУРА ПРОГРАММНОГО КОМПЛЕКСА РАСЧЕТА СПЕЦИАЛЬНЫХ ФУНКЦИЙ ВОЛНОВЫХ КАТАСТРОФ

В статье изложены особенности архитектуры вычислительного ядра программного комплекса, созданного для численного расчета специальных функций волновых катастроф (СВК). Рассмотрены примененные технические решения, их предпосылки, произведен анализ полученной архитектуры и направления её улучшения. Описаны используемые структуры данных и система модулей, показан механизм многопоточных вычислений.

Ключевые слова: *СВК, специальные функции волновых катастроф, теория катастроф, численные методы, параллелизм, архитектура программного обеспечения.*

A.S. Kryukovsky
S.V. Rogachev

THE ARCHITECTURE OF THE SOFTWARE CALCULATION OF SPECIAL FUNCTIONS OF WAVE CATASTROPHES

This article shows the main architectural characteristics of developed core software for computing special functions of wave catastrophes (SWC). Technical solutions applied during development and their preconditions are described in the article. The results of software design are analyzed and some future directions of improvement are observed. The materials contain information about used data structures and modular system. A component making the system multithreaded is also observed in the article.

Keywords: *SWC, special functions of wave catastrophes, catastrophes theory, numerical methods, parallel computing, software architecture.*

Введение

В рамках построения специализированного программного обеспечения решается множество задач оптимизации не только на микро-, но и на макроуровне. Без продуманной архитектуры и возможности дальнейшего расширения программа в конце концов становится бесполезной, поскольку решает лишь набор частных задач и достаточно быстро выходит из эксплуатации. Модульная архитектура и набор продуманных интерфейсов, напротив, позволяют программе развиваться во времени, обзаводясь новым функционалом. Таким образом, программа начинает представлять собой не замкнутую вычислительную систему, а, скорее, экосистему, платформу, позволяющую строить все новые и новые функциональные модули, объединенные в систему единым набором интерфейсов. Пример монолитного программного комплекса вычисления СВК рассмотрен в [1].

Архитектура программного комплекса расчета специальных функций волновых катастроф [2–4] построена на основе модульного подхода: вычислительный функционал, относящийся к какой-либо особенности, присутствует в системе в виде отдельного программного модуля, реализующего ряд функций обобщенного интерфейса. Модуль органично встраивается в систему за счет механизма регистрации модулей и может быть вызван для расчета особенности как со стороны пользователей программного комплекса, через высокоуровневый текстовый интерфейс, основанный на JSON, так и напрямую, программно.

Специфика предметной области позволяет производить высокоуровневые алгоритмические оптимизации, которые, в отличие от микрооптимизаций, легко переносимы и вносят значительный вклад в общую производительность

[1] Доктор физико-математических наук, профессор, декан факультета ИСиКТ НОУ ВПО «Российский новый университет».

[2] Аспирант НОУ ВПО «Российский новый университет».

программы. В случае с расчетом специальных функций волновых катастроф, в силу примененного метода ОДУ [5–7], возможен параллелизм по данным, что позволяет значительно увеличить скорость решения задачи в многопроцессорных системах и вычислительных сетях.

Архитектура программного обеспечения должна разрабатываться таким образом, чтобы возможные оптимизации не давали побочных эффектов. Например, в случае использования однопроцессорных машин не должно быть дополнительных накладных расходов, связанных с поддержкой многопроцессорной обработки (создание вычислительных потоков, синхронизация и др.).

Программный комплекс разрабатывается в качестве дополнения к информационной системе «Волновые катастрофы в радиофизике, акустике и квантовой механике» [8–21].

Система модулей

Программный комплекс (далее – ПК) предоставляет набор интерфейсов, позволяющий организовать всю систему в виде взаимосвязанных модулей. Эти модули имеют четкую структуру. Рассмотрим в качестве примера модуль расчета особенности C_4 (о катастрофе C_4 см. [22]). В прилагаемом примере строки пронумерованы для удобства изложения – номера в начале строк не являются частью программы.

```
--
 1 #include <wavecat/cmplx_catastrophe.h>
 2 #include <wavecat/point_array.h>
 3 #include <lib/integration/cmplx_runge_kutta.h>
 4
 5 #include <math.h>
 6
 7 /*
 8  * Edge catastrophe Csub4. W = (V), gamma = 1.
 9  */
10
11 enum parameters {
12       LAMBDA_1 = 0,
13       LAMBDA_2,
14       LAMBDA_3,
15 };
16
17 enum components {
18       V = 0,
19       V1,
20       V2
21 };
22
23 enum variables {
24       K = 0,
25       Asub3,
26       Asub3dl1,
27       Asub3dl2
28 };
29
30 #define PARAM(num) (catastrophe->parameter[(num)].cur_value)
31 #define VAR(num) (catastrophe->variable[(num)].cur_value)
32
33 void cmplx_catastrophe_Asub3_function(
34             const cmplx_catastrophe_t *const catastrophe,
35             const double t, const double complex *y,
36             double complex *const f);
37
38 void cmplx_catastrophe_Csub4_function(
39             const cmplx_catastrophe_t *const catastrophe,
40             const double t, const double complex *y,
41             double complex *const f)
42 {
43       double l1, l2, l3;
44       double complex U01, U02, U03, U12;
45
46       l1 = PARAM(LAMBDA_1) * t;
47       l2 = PARAM(LAMBDA_2) * t;
48       l3 = PARAM(LAMBDA_3) * t;
49
50       U01 = I * (I * VAR(Asub3) - l3 * y[V]);
51       U02 = I * VAR(Asub3dl1) - l3 * U01;
52       U12 = I * (I * VAR(Asub3dl2) - l3 * U02);
53       U03 = I * (4.0 * VAR(K) * U12 + 2.0 * I * l2 * U01 - l1 * y[V]);
54
```

```
55        f[V] =  PARAM(LAMBDA_1) * U01 +
56                PARAM(LAMBDA_2) * U02 +
57                PARAM(LAMBDA_3) * U03;
58  }
59
60  static void calculate(cmplx_catastrophe_t *const catastrophe,
61                const unsigned int i, const unsigned int j)
62  {
63        cmplx_equation_t *equation;
64        point_array_t *point_array;
65
66        /* Gamma function's precalculated values */
67        const double g14 = 3.625609908;
68        const double g34 = 1.225416702;
69
70        double module;
71        double phase;
72
73        assert(catastrophe);
74
75        equation = catastrophe->equation;
76        point_array = catastrophe->point_array;
77
78        assert(equation);
79        assert(point_array);
80
81        /* Calculate Pearcey function */
82        equation->initial_vector[V] = 0.5 * g14 * cexp(I * M_PI / 8.0);
83        equation->initial_vector[V1] = 0;
84        equation->initial_vector[V2] = 0.5 * I * g34 *
85              cexp(I * 3.0 * M_PI / 8.0);
86        equation_set_function(equation, cmplx_catastrophe_Asub3_function);
87        cmplx_runge_kutta(0.0, 1.0, 0.01, catastrophe);
88        VAR(Asub3) = equation->resulting_vector[V];
89        VAR(Asub3dl1) = equation->resulting_vector[V1];
90        VAR(Asub3dl2) = equation->resulting_vector[V2];
91
92        /* Calculate Csub4 */
93        equation->initial_vector[V] = M_PI;
94        equation_set_function(equation, cmplx_catastrophe_Csub4_function);
95        cmplx_runge_kutta(0.0, 1.0, 0.01, catastrophe);
96
97        module = cabs(equation->resulting_vector[V]);
98        phase = (180.0 / M_PI) * carg(equation->resulting_vector[V]);
99
100       point_array->array[i][j].module = module;
101       point_array->array[i][j].phase = phase;
102 }
103
104 cmplx_catastrophe_t *cmplx_catastrophe_Csub4_fabric(
105       const parameter_t *const parameter,
106       const int num_parameters, const cmplx_variable_t *const variable,
107       const int num_variables)
108 {
109       return cmplx_catastrophe_fabric(parameter, num_parameters, variable,
110                   num_variables, 3, "Csub4", calculate,
111                   cmplx_catastrophe_Csub4_function);
112 }
113
114 static cmplx_catastrophe_desc_t cmplx_catastrophe_Csub4_desc = {
115       .sym_name = "Csub4",
116       .fabric = cmplx_catastrophe_Csub4_fabric,
117       .num_parameters = 3,
118       .num_variables = 4
119 };
120
121 static int cmplx_catastrophe_Csub4_init(void) __attribute__ ((constructor));
122 static int cmplx_catastrophe_Csub4_init(void)
123 {
124       fprintf(stderr, "Catastrophe Csub4 (complex) initialization.\n");
125       register_cmplx_catastrophe_desc(&cmplx_catastrophe_Csub4_desc);
126 }
```

--

Листинг 1. Модуль вычисления СВК C_4

Каждый модуль, осуществляющий вычисление СВК, должен быть зарегистрирован в системе. Регистрацию осуществляет функция *register_cmplx_catastrophe_desc*. На вход функции передается дескриптор модуля, включающий в себя информацию о названии СВК, количестве ее параметров, количестве переменных (под переменной в данном случае понимается не переменная языка программирования, а специальный объект, служащий для передачи параметров внутри функций модуля) и фабричном методе объекта катастрофы.

Несмотря на то что язык Си не поддерживает абстракции классов и объектов, а также наследование, интерфейс системы имеет объектно ориентированную архитектуру. Для выполнения расчетов для выбранной особенности создается объект катастрофы, который может быть вычислен. За конструирование объекта отвечает фабрика, представленная в дескрипторе модуля. Легко заметить, что эта фабрика является ни чем иным, как оберткой над обобщенной функцией *cmplx_catastrophe_fabric*, выполняющей всю работу по конструированию объекта.

Каждый модуль имеет свою реализацию функции *calculate*, представленной в объекте катастрофы. Эта функция отвечает за вычисления и вызывается системой для каждой точки, и, соответственно, для всех сочетаний значений параметров. В примере реализация этой функции находится в строках 60–102. В строках 81–86 подготавливается система уравнений для расчета функции Пирси, в строке 87 выполняется расчет, далее происходит сохранение промежуточных результатов в объектах-переменных. Объект уравнения переиспользуется для вычисления функции C_4. Для этого требуется его повторная инициализация и запуск вычислений. Расчет самой особенности выполняется в функции *cmplx_catastrophe_Csub4_function*, код которой неявно вызывается подпрограммой интегрирования через механизм обратных вызовов (в западной литературе – callback). Строки 50–53 следуют из системы обыкновенных дифференциальных уравнений для особенности C_4, далее выполняется вычисление полного приращения функции.

Механизм внутренних взаимодействий между компонентами системы скрыт от глаз разработчика. Конструирование промежуточных объектов, передача данных между функциями, вычисление новых значений параметров – все это происходит косвенно и не требует никаких лишних действий.

Каждый модуль особенности, зарегистрированный в системе, может быть найден по имени этой особенности. Поиск дескриптора нужного модуля осуществляется системой при помощи вызова *find_cmplx_catastrophe_desc*, имеющего следующую реализацию.

```
 1 cmplx_catastrophe_desc_t *find_cmplx_catastrophe_desc(const char *sym_name)
 2 {
 3       cmplx_catastrophe_desc_t *cd;
 4       list_head_t *pos;
 5
 6       list_for_each(pos, &cmplx_catastrophe_desc_list) {
 7             cd = list_entry(pos, cmplx_catastrophe_desc_t, list);
 8             if (strcmp(cd->sym_name, sym_name) == 0)
 9                   return cd;
10       }
11
12       return NULL;
13 }
```

Листинг 2. Поиск дескриптора особенности

Дескрипторы особенностей организованы в список. Поскольку количество особенностей не так велико и поиск осуществляется лишь раз за время обработки задания, нет необходимости дополнительной алгоритмической оптимизации и использования более сложной структуры данных, например дерева.

Для построения задания требуется пользовательский ввод, указывающий, какую специальную функцию необходимо вычислить, в каких диапазонах будут изменяться параметры, каковы начальные значения переменных. Ввод предоставляется программе в переносимом текстовом формате JSON. Обработка данных в этом формате осуществляется отдельной библиотекой, предоставляющей системе вызов *json_input*. Эта функция принимает на вход строку с описанием задания, после разбора входной строки строится объект особенности и выполняются вычисления. Результат вычислений попадает на стандартный вывод в текстовом представлении, также являющемся объектом JSON.

Строгая организация внутренних интерфейсов и шаблонность вычислительных модулей позволяет генерировать их программно. Благодаря этому свойству удалось спроектировать предметно ориентированный язык [23], скрывающий большинство сложностей, характерных для языка программирования Си. Транслятор для этого

предметно ориентированного языка достаточно прост и может быть реализован на любом современном ЯВУ без применения специальных средств (разбор грамматики легко осуществляется любым нисходящим методом). Пробный транслятор для языка реализован на ЯВУ Ocaml и занимает чуть более 300 строк кода.

Многопоточные вычисления

Для реализации многопоточных вычислений не потребовалось каких-либо серьезных изменений в архитектуре приложения. Одна из причин – простота декомпозиции задачи в силу ее специфики – задача полностью независима по данным, что сказывается на простоте реализации параллельного алгоритма.

Другая причина кроется, собственно, в архитектурном решении: для поддержки различных видов параллелизма предусмотрен механизм, позволяющий подменять основной цикл вычислений на прокси-функцию, осуществляющую декомпозицию задачи.

Параллелизм привнесен в систему в виде отдельной подсистемы. Подсистема представляет собой промежуточный слой, транслирующий одно задание в набор заданий для вычислительных потоков, которые создаются им автоматически. Результат вычислений также автоматически агрегируется, как только частные результаты будут готовы.

Вместо основного цикла вычислений *cmplx_catastrophe_loop*, который реализован в базовой системе, происходит вызов функции *cmplx_catastrophe_parallel_loop*, который и осуществляет основную работу по подготовке заданий, запуску и остановке вычислительных потоков, сбору результатов вычислений.

```
--
 1       thread_idx = cores;
 2       while (thread_idx--) {
 3             catastrophe->parameter[p1_idx].min_value = p1_min;
 4             p1_max = p1_step * (steps_per_core - 1) + p1_min;
 5             catastrophe->parameter[p1_idx].max_value = p1_max;
 6             catastrophe->parameter[p1_idx].num_steps = steps_per_core;
 7             p1_min = p1_max + p1_step;
 8
 9             new_cat = catastrophe_desc->fabric(
10                         catastrophe->parameter,
11                         catastrophe->num_parameters,
12                         catastrophe->variable,
13                         catastrophe->num_variables);
14
15             if (!new_cat) {
16                   WAVECAT_ERROR(-1);
17                   goto fail;
18             }
19
20             tcatastrophe[thread_idx] = new_cat;
21
22             res = pthread_create(&thread[thread_idx], NULL,
23                         cmplx_parallel_loop_thread,
24                         (void *) new_cat);
25             if (res) {
26                   WAVECAT_ERROR(-1);
27                   goto fail;
28             }
29       }
30
31       thread_idx = cores;
32       first_idx = 0;
33       while (thread_idx--) {
34             void *retval = NULL;
35
36             res = pthread_join(thread[thread_idx], &retval);
37             if (res) {
38                   WAVECAT_ERROR(-1);
39                   goto fail;
40             }
41
42             copy_part_point_array(catastrophe->point_array,
43                         tcatastrophe[thread_idx]->point_array,
44                         first_idx);
45             first_idx +=
46                   tcatastrophe[thread_idx]->point_array->num_steps_x;
47
48             destruct_cmplx_catastrophe(tcatastrophe[thread_idx]);
49       }
--
```

Листинг 3. Основной код, осуществляющий декомпозицию задачи

Цикл 2–29 осуществляет декомпозицию задачи по данным, выполняется вычисление границ изменения параметров, конструируются объекты особенностей, запускаются потоки. Цикл 33–49 осуществляет ожидание завершения подзадач, собирает результаты вычислений и производит их обратную композицию.

Часть кода не приведена из соображения экономии места. Например, в приведенных листингах порой отсутствуют необязательные для понимания части кода, ответственные за блокировки, обработку ошибок, освобождение памяти и деинициализацию.

За счет механизма подмены основного вычислительного цикла становится возможным полностью сократить издержки на управление потоками и декомпозицию задачи в случае использования однопроцессорной вычислительной машины. Так, вместо цикла для параллельной обработки будет вызван базовый вычислительный цикл.

Пример построения

В качестве примера СВК, рассчитанной системой, рассмотрим краевую особенность B_3.

Особенность можно представить в виде интеграла:

$$V_{B_3}(\lambda_1, \lambda_2) = \int_0^{+\infty} \exp(i(kx^3 + \lambda_2 x^2 + \lambda_1 x))dx, k = \pm 1. \quad (1)$$

Фундаментальный вектор включает саму функцию и ее первую производную:

$$W = (V, V^1). \quad (2)$$

В качестве начального выберем нулевой вектор:

$$S^0 = (0, 0). \quad (3)$$

Компоненты правой части системы ОДУ выражаются следующим образом:

$$\begin{aligned} &\frac{\partial V}{\partial \lambda_1} = V^1, \\ &\frac{\partial V}{\partial \lambda_2} = iU_{11}, \\ &\frac{\partial V^1}{\partial \lambda_1} = \frac{k}{3}(\lambda_1 V - 2i\lambda_2 V^1 - i) \equiv U_{11}, \\ &\frac{\partial V^1}{\partial \lambda_2} = \frac{ik}{3}(V + \lambda_1 V^1 - 2i\lambda_2 U_{11}). \end{aligned} \quad (4)$$

Начальные значения выражаются через гамма-функцию:

$$\begin{aligned} V(0) &= \frac{1}{3}\Gamma(\frac{1}{3})\exp(\frac{ik\pi}{6}), \\ V^1(0) &= \frac{i}{3}\Gamma(\frac{2}{3})\exp(\frac{ik\pi}{3}). \end{aligned} \quad (5)$$

Результат вычислений, полученный от вычислительного ядра в виде объекта JSON, может быть представлен в виде двух контурных графиков:

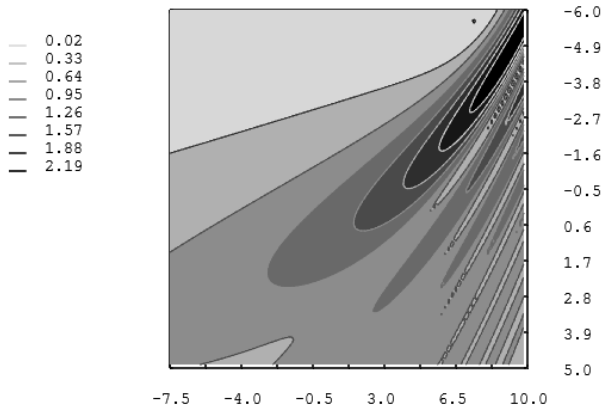

Рис. 1. Краевая катастрофа B_3 (амплитуда), $\lambda_1 = [-7.5, 10]$, $\lambda_1 = [-6,5]$, $k = 1$

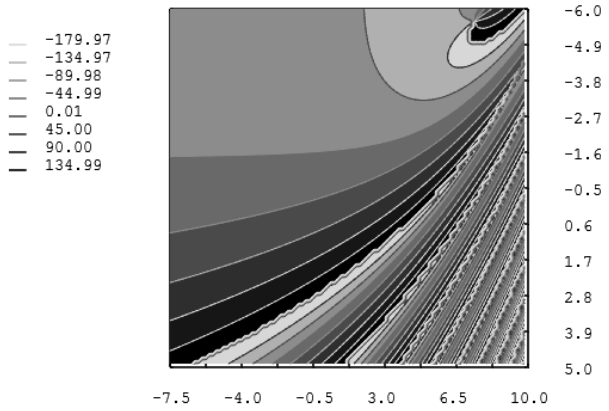

Рис. 2. Краевая катастрофа B_3 (фаза), $\lambda_1 = [-7.5, 10]$, $\lambda_1 = [-6,5]$, $k = 1$

Заключение

Рассмотренная архитектура не лишена недостатков. Например, реализация функции *calculate* во многих случаях может быть оптимизирована. Многократное вычисление начальных параметров задачи при каждом вызове функции приводит к падению производительности. Легко заметить, что во многих особенностях эти параметры постоянны (это чаще всего характерно для основных катастроф). В таком случае более логичной была бы отдельная функция инициализации, выполняемая лишь раз за время вычислений. С другой стороны, для ряда особенностей это заключение неверно, поскольку начальными значениями становятся результаты вычисления других функций. Разумным выходом из ситуации видится применение переменной флагов, хранящейся в дескрипторе особенности. Подоб-

ная переменная поможет отмечать те или иные специфические состояния модуля расчета особенности, например необходимость пересчета ряда значений.

Оптимизация функции *calculate* в каждом модуле, а также ряд микрооптимизаций в функциях интегрирования может привести к колоссальному росту производительности: по результатам профилирования именно эти функции вызываются чаще всего. Это также можно проследить, анализируя исходный код вручную.

Подсистема параллельных вычислений нуждается в доработке: достаточно большое время тратится на создание и уничтожение вычислительных потоков – это достаточно дорогая операция. Решить проблему дополнительных расходов по времени возможно путем использования пула потоков. Пул потоков представляет собой механизм, организующий вычислительные потоки по признакам «используемый» и «неиспользуемый». Предоставляемый интерфейс позволяет выбрать один неиспользуемый поток из набора заранее запущенных и выполнить в его контексте поступившую задачу. После окончания вычислений поток возвращается в пул и может быть выделен для решения других задач. Если потоков не хватает, чтобы обслужить все поступившие запросы, запросы выстраиваются в очередь в ожидании обработки.

Литература

1. Дорохина Т.В., Крюковский А.С., Малышенко А.Б. Разработка численных алгоритмов расчета и визуализации волновых катастроф // Вестник Российского нового университета. – Серия «Управление, вычислительная техника и информатика». – 2008. – Вып. 3. – С. 25–48.

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 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. –261 с. ISSN 2304-0297 (CD-ROM). – С. 139–144.

4. Рогачев С.В., Крюковский А.С. Специализированная система расчета и визуализации специальных функций волновых катастроф // Труды XIV Международной научной конференции «Цивилизация знаний: проблемы и смыслы образования» (Москва, 26–27 апреля 2013 г.). – М. : РосНОУ, 2013. – Ч. 2. – С. 171–174.

5. 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.

6. Крюковский А.С. Равномерная асимптотическая теория краевых и угловых волновых катастроф : монография. – М. : РосНОУ, 2013. – 368 с.

7. Крюковский А.С., Лукин Д.С. Теория расчета эталонных фокальных и дифракционных электромагнитных полей на основе специальных функций волновых катастроф // Радиотехника и электроника. – 2003. – Т. 48. – № 8. – С. 912–921.

8. Дорохина Т.В., Крюковский А.С., Растягаев Д.В. Принципы разработки мультимедийной информационной системы «Волновые катастрофы в радиофизике, акустике и квантовой механике» // Россия: перспективы прорыва в цивилизацию знаний. Информационные системы и компьютерные технологии: тезисы докл. V межвузовской научной конференции Российского нового университета, 16–17.04.2004 г. – М. : РосНОУ, 2004. – С. 291–293.

9. Дорохина Т.В., Крюковский А.С., Растягаев Д.В. Мультимедийная информационная система «Волновые катастрофы в радиофизике, акустике и квантовой механике» // Труды VI Международного симпозиума «Интеллектуальные системы» (INTELS'2004), Россия, Саратов, СГТУ, 29.06–2.07.2004 г. – М. : РУСАКИ, 2004. – С. 469–470.

10. Дорохина Т.В., Крюковский А.С., Растягаев Д.В. Мультимедийная информационная система «Волновые катастрофы в радиофизике, акустике и квантовой механике» // Вестник Российского нового университета. – Серия «Естествознание, математика, информатика». – 2004. – Вып. 4. – С. 83–86.

11. Гришин В.С., Дорохина Т.В., Крюковский А.С., Растягаев Д.В. Разработка электронного справочника и компьютерное моделирование волновых катастроф // Цивилизация знаний: будущее и современность : материалы Всероссийской научной конференции (Москва, май 2005). – М. : РосНОУ, 2005. – С. 144–147.

12. Дорохина Т.В., Ипатов Е.Б., Крюковский А.С., Лукин Д.С., Палкин Е.А., Растягаев Д.В. Мате-

матическое компьютерное моделирование волновых полей типа катастроф // Распространение радиоволн : сборник докладов XXI Всероссийской научной конференции (Йошкар-Ола, 25–27 мая 2005 г.). – Йошкар-Ола : МарГТУ, 2005. – Т. 2. – С. 336–339.

13. Дорохина Т.В., Крюковский А.С., Лукин Д.С., Палкин Е.А., Растягаев Д.В. Математическое моделирование волновых полей типа катастроф с помощью специализированной информационно-справочной системы : труды Российского научно-технического общества радиотехники, электроники и связи им. А.С. Попова. – Серия: научная сессия, посвященная Дню радио 17–18.05.2006 г. – Выпуск LXI. – М. : РНТОРЭС, 2006. – С. 287–289.

14. Дорохина Т.В., Крюковский А.С., Гришин В.С. Применение специализированной информационной системы для компьютерного моделирования волновых полей // Цивилизация знаний: российские реалии : труды Седьмой Всероссийской научной конференции (Москва, 22 апреля 2006). – М. : РосНОУ, 2006. – Ч. I. – С. 198–200.

15. Дорохина Т.В., Крюковский А.С., Лукин Д.С. Информационная система «Волновые катастрофы в радиофизике, акустике и квантовой механике» // Электромагнитные волны и электронные системы. – 2007. – Т. 12. – № 8. – С. 71–75.

16. Дорохина Т.В., Крюковский А.С., Лукин Д.С., Волкова Е.В., Костью А.О., Павлова М.В. Создание информационной системы волновой теории катастроф и её применение при математическом моделировании // Вестник Российского нового университета. – 2007. – Выпуск 2. – С. 91–107.

17. Дорохина Т.В.,Крюковский А.С.,Растягаев Д.В. Информационные технологии представления катастроф в волновой физике // Материалы XV межрегиональной научно-технической конференции «Обработка сигналов в системах наземной радиосвязи и оповещения» Московского и Нижегородского отделений НТОРЭС им. А.С. Попова (Н.-Новгород, октябрь 2007 г.). – Н. Новгород – Москва : НТОРЭС, 2007. – С. 329–331.

18. Волкова Е.В., Дорохина Т.В., Крюковский А.С. Разработка программного модуля построения амплитудно-фазовых сечений волновых катастроф для мультимедийного справочника «Волновые катастрофы в радиофизике, акустике и квантовой механике» // Цивилизация знаний: российские реалии : труды Восьмой Всероссийской научной конференции (Москва, 20–21 апреля 2007). – М. : РосНОУ, 2007. – Ч. I. – С. 109–111.

19. Дорохина Т.В., Крюковский А.С., Лукин Д.С. Принципы проектирования и построения информационно-справочной системы волновой теории катастроф // Цивилизация знаний: российские реалии : труды Восьмой Всероссийской научной конференции (Москва, 20–21 апреля 2007). – М. : РосНОУ, 2007. – Ч. I. – С. 158–164.

20. Волкова Е.В., Дорохина Т.В., Крюковский А.С. Построение амплитудно-фазовых сечений волновых катастроф в информационной системе «Волновые катастрофы в радиофизике, акустике и квантовой механике» // Труды Российского научно-технического общества радиотехники, электроники и связи им. А.С. Попова. – Серия: научная сессия, посвященная Дню радио 14–15.05.2008 г. – Выпуск LXIII. – М. : РНТОРЭС, 2008. – С. 293–295.

21. Дорохина Т.В., Крюковский А.С., Павлова М.В. Представление лучевых и каустических структур волновых катастроф и схем их подчинения в информационно-справочной системе // Цивилизация знаний: российские реалии : труды Восьмой Всероссийской научной конференции (Москва, 20–21 апреля 2007). – М. : РосНОУ, 2007. – Ч. III. – С. 130–136.

22. Крюковский А.С., Лукин Д.С., Палкин Е.А. Применение теории краевых катастроф для построения равномерных асимптотик быстроосциллирующих интегралов // Дифракция и распространение волн : междувед. сборник МФТИ. – М., 1985. – С. 4–21.

23. Рогачев С.В. Проектирование предметно-ориентированного языка для расчета специальных функций волновых катастроф // Вестник Российского нового университета. – Серия «Управление, вычислительная техника и информатика». – 2013. – Вып. 4. – С. 47–52.