Санкт-Петербургский государственный университет
**Метаматико-Механический Факультет**
**Кафедра Системного Программирования**

# Теодам Александр Андреевич

## Курсовая работа

# Анализ распределенной вычислительной сети "RuSolver".

Направление 02.04.03
Математическое Обеспечение и
Администрирование Информационных Систем

Научный руководитель,
доктор физ.-мат. наук,
профессор Терехов А. Н.

Заведующий кафедрой,
доктор физ.-мат. наук,
профессор Терехов А. Н.

Санкт-Петербург
2016

# ВВЕДЕНИЕ

Создание компьютеров, как и многих других революционных изобретений человечества, в первую очередь было вызвано необходимостью повысить качество и производительность работы военных и разведывательных структур. Автоматизация вычислений позволила упростить решение таких сложных математические задач, как расшифровка перехваченных сообщений противника, задач расчёта скорости движения цели и вероятность поражения цели.

По мере своего развития и роста вычислительной мощи компьютеры стали приносить пользу в промышленности и бизнесе. Данное событие определило новый этап в развитии вычислительной техники — её эволюция стала подчиняться закону Гордона Мура, в котором говорится, что число транзисторов на кристалле микропроцессора (вычислительная мощность компьютера) удваивается примерно каждые два года.

В последнее время, чтобы получить возможность задействовать на практике ту дополнительную вычислительную мощность, которую даёт закон Мура, стало необходимо задействовать параллельные вычисления. На протяжении многих лет, производители процессоров постоянно увеличивали тактовую частоту и параллелизм на уровне инструкций, так что на новых процессорах старые однопоточные приложения исполнялись быстрее без каких-либо изменений в программном коде. Сейчас по разным причинам производители процессоров предпочитают многоядерные архитектуры, и для получения всей выгоды от возросшей производительности центрального процессора программы должны переписываться в соответствующей манере.

Понятие параллельное программирование означает достаточно широкую область, которая связана с организацией расчётов на вычислительных системах, включающих в себя несколько процессорных устройств. К таким системам относятся многоядерные процессоры, многопроцессорные машины

с общей памятью, высокопроизводительные вычислительные кластеры с распределенной памятью или гибридной архитектурой [1].

Параллельным вычислениям в последнее время уделяется большое внимание. Это связано главным образом с двумя факторами. Первый фактор обусловлен научно-техническим прогрессом, в результате которого появились новые области знаний, требующие применения методов математического моделирования. Сами модели также существенно усложнились. В итоге происходит неуклонное возрастание потребности в ресурсоёмких расчётах, которые в ряде случаев можно выполнить только на базе высокопроизводительной техники с помощью методов параллельных или распределённых вычислений.

Другой существенный фактор, в результате которого интерес к параллельным вычислениям существенно вырос, состоит в повсеместном распространении параллельных компьютеров. В последнее время многопроцессорные серверы можно часто встретить на средних и крупных предприятиях, в банках, исследовательских институтах и центрах. В связи с появлением многоядерных процессоров многие пользователи стали обладателями мини-суперкомпьютеров на своих рабочих местах.

Существенный прогресс в области сетевых технологий позволил использовать для параллельных вычислений сети предприятий, компьютеры учебных классов, объединённые как в локальные сети, так и в составные. Появилась возможность создания дешёвых вычислительных кластеров.

В итоге, можно с уверенностью утверждать, что параллельные информационные технологии превратились из узконаправленной дисциплины в необходимую составляющую комплекса знаний разработчика современного программного обеспечения.

# 1. АНАЛИЗ ИНФОРМАЦИОННОЙ СИСТЕМЫ RuSolver

В настоящее время для распараллеливания вычислений в условиях составных сетей используются соответствующие программные оболочки, реализованные в архитектуре клиент-сервер. Среди этих программных решений распараллеливания можно выделить следующие системы:

- Система добровольных вычислений Rosetta@Home, использующая часть вычислительных ресурсов машины-клиента для научных расчётов.
- Система Distributed Internet MEasurements & Simulations (DIMES), занимающаяся построением подробной карты интернета. Клиент данной системы выполняет измерения состояния сети, такие как *traceroute* или *ping*. DIMES позволяет наблюдать в реальном времени «живую» карту интернета, перестраивающуюся с учётом свежеполученной информации.
- Проект RainbowCrack, создающий базу по всем возможным паролям и соответствующим им хэшам, а также предоставляющий доступ к этой базе. Сейчас в проекте созданы сотни гигабайт всех возможных паролей, которые позволяют с вероятностью порядка 99% найти за несколько минут обратное преобразование из хэша в любой пароль длиной до 7 символов (используются буквы, цифры, и многие спецсимволы). В базе RainbowCrack уже имеются пароли, зашифрованные по алгоритмам LanMan (авторизация в Windows), NT LanMan (авторизация в Windows NT, в том числе сетевых доменах), MD2, MD4, MD5, SHA1, RIPEMD-160, MySQL 3.23, MySQL SHA1.
- Проект RuSolver, предоставляющий услуги по восстановлению паролей, расшифровки хэшей и решению прикладных задач пользователя.

В каждой из этих систем есть свои как достоинства, так и недостатки, но наибольший интерес с точки зрения участия пользователя в развитии системы представляет система RuSolver, к достоинствам которой относятся:

- объединение различных операционных систем и физических устройств как стационарных, так и мобильных;
- не только решение задач, определённых услугами системы, но и встраивание собственных решений для ещё не предоставленных услуг;
- возможность задействовать в решении прикладной задачи собственные вычислительные мощности.

С учётом отмеченных положительных сторон многопоточной гетерогенной информационной системы RuSolver в дальнейшей работе для реализации прикладных задач будем использовать её среду. Для этого рассмотрим информационную систему RuSolver более подробно.

Информационная система RuSolver представляет собой программную оболочку, позволяющую распараллеливать решение задач пользователей.

Взаимодействие системы RuSolver с вычислительными узлами построено на базе клиент-серверной архитектуры (рисунок 1.1). Сетевая архитектура, в которой задачи и нагрузка распределяются между сервером и клиентами, позволяет объединять различные операционные системы и физические устройства, а также своевременно обновлять программное обеспечение [2].

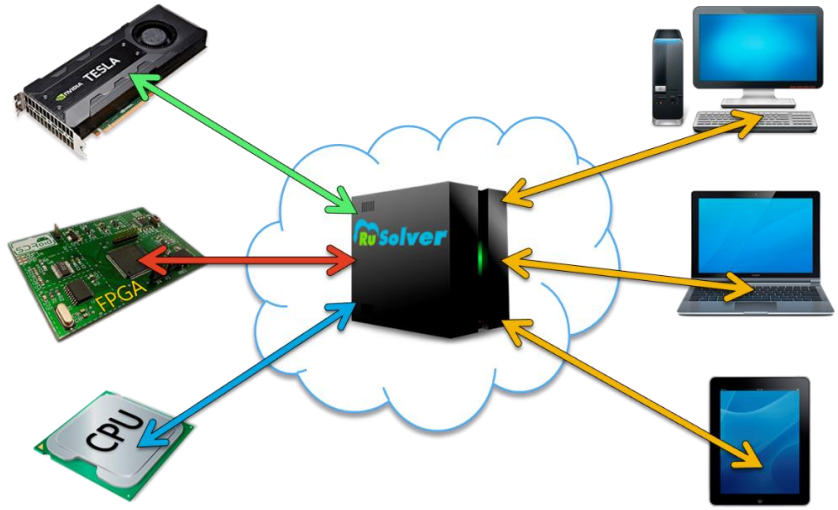


Рисунок 1.1 — Клиент-серверная архитектура системы RuSolver

Одной из главных особенностей системы RuSolver является поддержка встраивания пользовательских реализаций решения задач. Решение одной задачи представляется динамически подключаемой библиотекой.

## 1.1 Взаимодействие системы RuSolver с библиотеками реализаций решения прикладных задач на устройстве

Реализация алгоритма решения пользовательской задачи на устройстве осуществляется путём динамической подключаемой библиотеки (далее DLL-библиотеки).

DLL-библиотека является основной частью системы RuSolver. Она выполняет функцию идентификации исходных данных, а также подготовки, распределения и решения пользовательской задачи.

Формализация:

$$D^k_i = \{ D^k_1, D^k_2, ..., D^k_{Nk} \},$$

где $D^k$ — множество DLL-библиотек программной реализации k-й задачи пользователя для устройств;

I — DLL-библиотека k-й задачи ($i = 1, 2, ..., N_k$), $k = 1, 2, ..., M$;

$N_k$ — количество DLL-библиотек k-й задачи, не большее количества устройств;

M — количество решаемых задач пользователя.

DLL-библиотека создаётся для конкретного типа устройств (центральный процессор, устройство NVIDIA CUDA, программируемая логическая интегральная схема) и должна быть построена в соответствии с установленным стандартом, который предъявляет необходимые требования к составу библиотеки [2]:

1. Константы:
    1.1. *Platform* — версия платформы;
    1.2. *Group* — идентификатор группы пользовательской задачи;
    1.3. *Name* — строка наименования пользовательской задачи (8 символов в ANSI кодировке);
    1.4. *Caption* — строка заголовка пользовательской задачи (32 символа в ANSI кодировке);
    1.5. *Hint* — строка описания пользовательской задачи (512 символов в ANSI кодировке);
    1.6. *JClass* — класс пользовательской задачи.
2. Функции:
    2.1. *Parser* — функция анализа пользовательской задачи и формирования данных на основе этого анализа;
    2.2. *Cut* — функция распределения данных пользовательской задачи;
    2.3. *Solve* — функция алгоритма решения пользовательской задачи;
    2.4. *Result* — функция формирования результата пользовательской задачи.

Схема процесса решения прикладной задачи с помощью оболочки RuSolver представлена на рисунке 1.2.

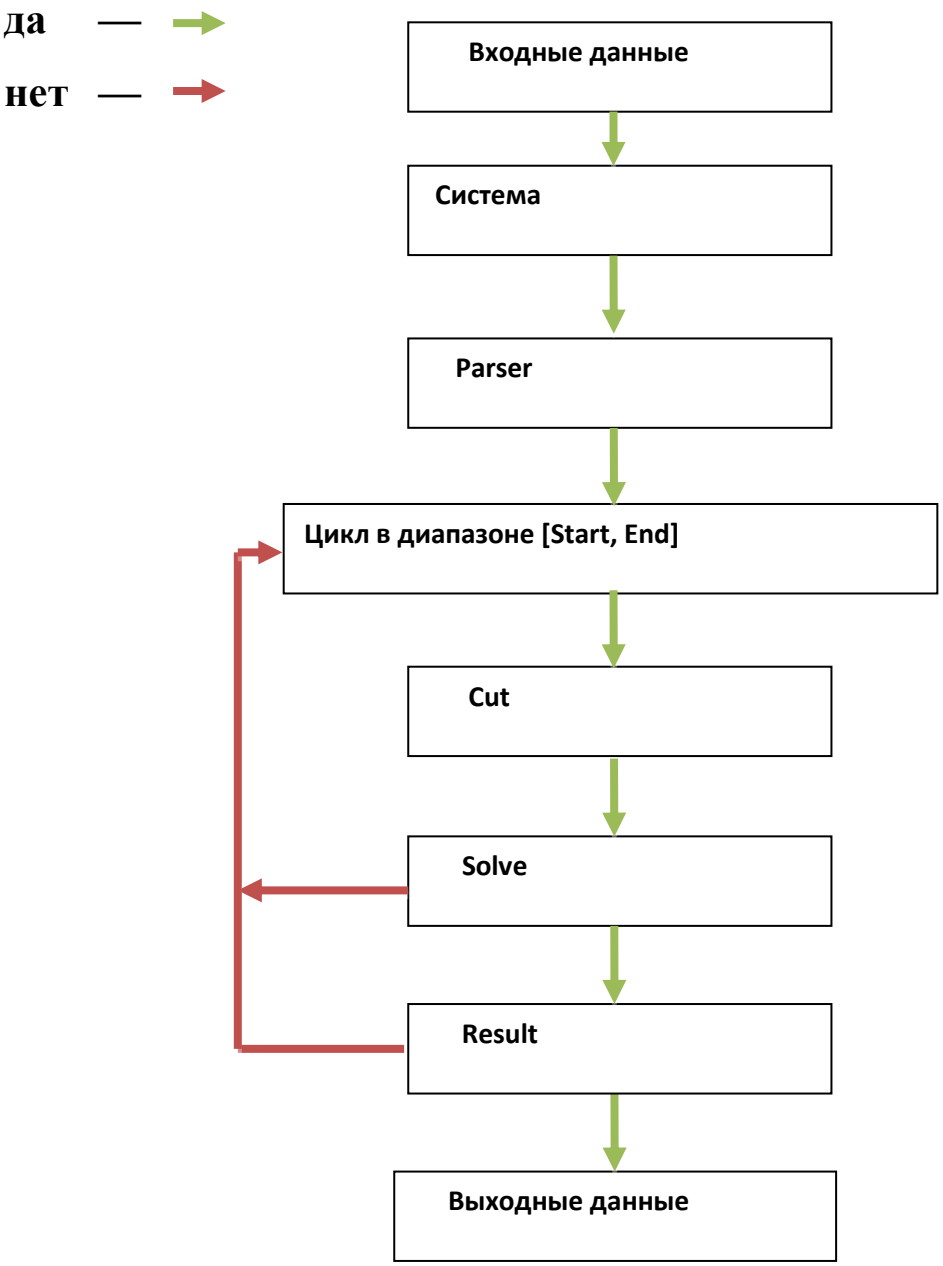


Рисунок 1.2 — Схема процесса решения прикладной задачи

На рисунке 1.3 показана часть алгоритмов, которую должен реализовать разработчик DLL-библиотеки, и часть алгоритмов, которую выполняет оболочка RuSolver.

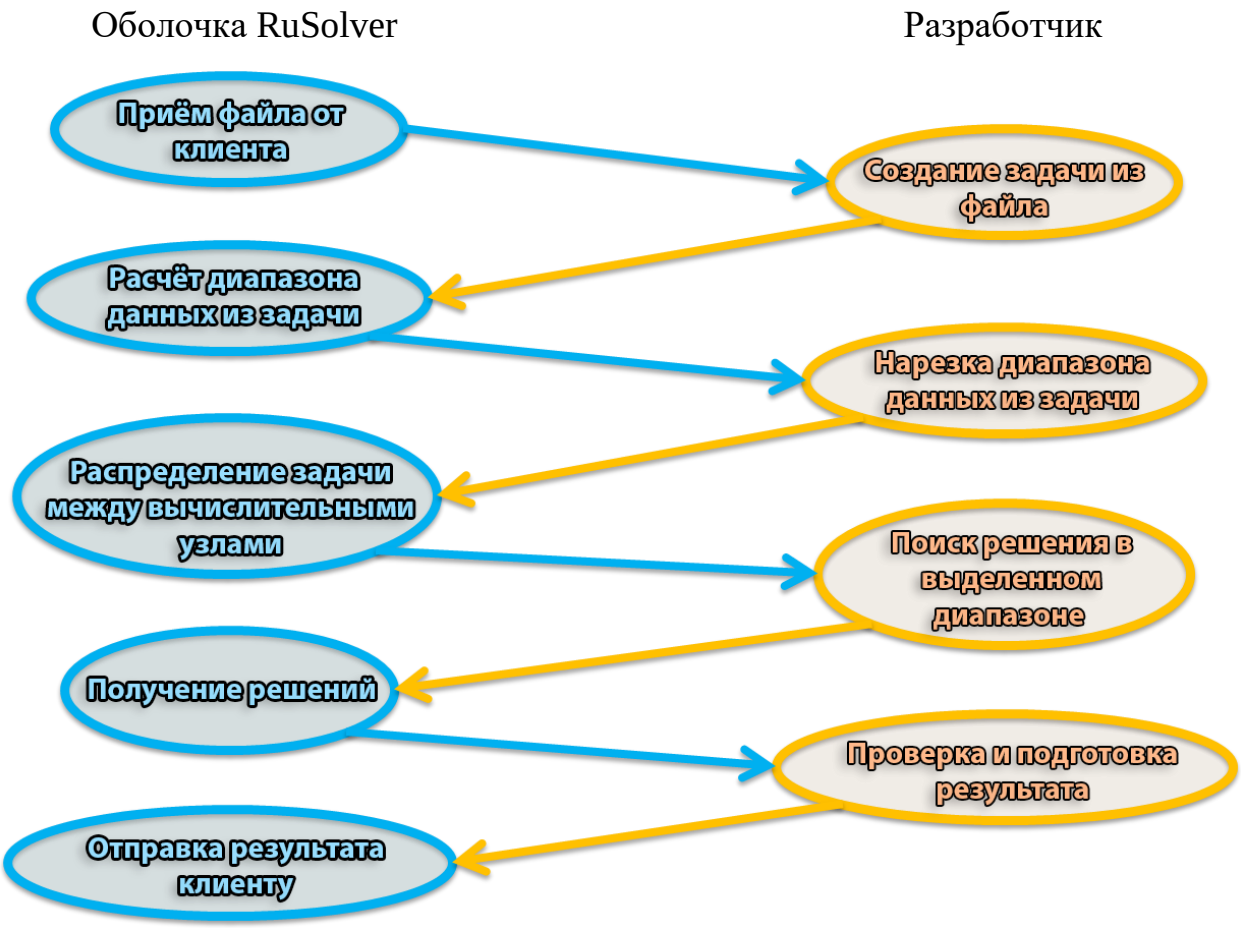


Рисунок 1.3 — Взаимодействие оболочки RuSolver с функциями из DLL-библиотеки разработчика

## 1.2 Классификация решаемых задач

DLL-библиотека предполагает собой решение таких пользовательских задач, поиск решения в которых может быть значительно ускорен за счёт использования распределения.

DLL-библиотека должна быть написана на языке программирования C++, поэтому здесь и далее, при описании структурных единиц библиотеки и их состава, будем использовать понятия и термины из этого языка.

Объём пользовательской задачи в системе определяется диапазоном от ноля до максимального количества всех возможных результатов итераций. Максимальное количество итераций определяется структурой типа *TStep*, ко-

торая предназначена для хранения больших чисел. Максимально допустимым значением для этих чисел является $2^{512}$.

Конечный результат может определяться на основе текущей итерации или с помощью заранее определённого диапазона, выраженного структурой типа *TDistance*, которая описывает диапазон возможных результатов.

Для определения минимальной единицы решения задачи разработчику DLL-библиотеки требуется определить шаблон. Назовём процесс минимального решения — элементарной операцией данного шаблона задач. Шаблон определяется константой *JClass*, которая может принимать ряд различных значений, характеризующих тип шаблона.

### 1.2.1 Неопределённый шаблон

Неопределённый шаблон представлен константой *JClass*, равной нолю. Разработчику рекомендуется использовать данный шаблон, если не один из других не подходит для его задачи. В этом случае конечное число элементарных операций, необходимое для решения задачи, определяется разработчиком DLL-библиотеки после выполнения функции анализа и формирования данных для задачи.

В неопределённом шаблоне пользовательские настройки диапазона отсутствуют, а структура *TDistance* не используется.

### 1.2.2 Символьный шаблон

Символьный шаблон представлен константой *JClass*, равной числу 3. Данный шаблон используется для нахождения символьной последовательности требующейся для решения задачи. За одну элементарную операцию будет принято решение для одной символьной последовательности. Конечное количество элементарных операций определяется автоматически ядром системы по максимальному количеству символьных последовательностей, которые возможно сформировать из структуры диапазона *TDistance*. Далее поиск ведётся по установленному разработчиком набору символов и длин по-

следовательностей. Если они не установлены, то поиск ведётся по максимальному набору.

Пользователь может сократить диапазон поиска за счёт уменьшения количества символов в каждой позиции отдельно и длин последовательностей, а также может воспользоваться своим набором готовых последовательностей, представив их в файле формата системы или указав необходимые для решения символы на сайте проекта RuSolver [3].

## 1.3 Порядок работы с DLL-библиотекой

### 1.3.1 Анализ и формирование пользовательской задачи

Система RuSolver добавляет входные данные для задачи исключительно в виде файлов. Для их обработки вызывается функция *Parser*, в которую через параметр с именем *FileName* передаётся путь к файлу. Функция должна выполнять следующие операции [2]:

- Проверять корректность передаваемого файла.
- Формировать данные, требуемые для решения, и записывать их в поле с именем *Data*, находящееся в объекте структуры *TData*. В поле с именем *Size* должен быть записан размер данных. Также, при необходимости промежуточных данных, разработчик самостоятельно заполняет файл, путь к которому указан в поле с именем *File*.
- При выборе неопределённого шаблона задач необходимо задать значение переменной типа *TStep** с именем *End*, которая определяет максимальное количество элементарных операций для решения пользовательской задачи. В остальных случаях изменение переменной *End* не приведёт к изменению решаемого диапазона.

### 1.3.2 Распределение диапазона

Для распределения вычислений на различных устройствах входные данные задачи разбивается на части с помощью функции *Cut*. Для правильного решения всей задачи функция должна в соответствии с диапазоном

[*Start*; *Start* + *Segment*] и представленными входными данными задачи (параметр с именем *TaskSource*) подготовить данные для решения части задачи.

Подготовленные данные для решения задачи на диапазоне [*Start*; *Start* + *Segment*] необходимо представить в параметре с именем *TaskDestination*.

### 1.3.3 Решение задачи

Для решения задачи на диапазоне [*Start*; *Start* + *Segment*] над её данными, представленными параметром с именем *Task*, вызывается функция *Solve*. Значение параметра *Progress*, находящееся в диапазон [0, *Segment*], должно отражать текущий прогресс решения задачи и изменяться не реже одного раза в 10 секунд, иначе функция будет считаться некорректной, что приведёт к её принудительной остановке со стороны системы.

Для получения текущего значения параметра элементарной операции необходимо воспользоваться функцией с именем *GetKey* подав в неё значение параметра с именем *Distance*.

При получении каких-либо результатов их можно сохранить, не прерывая работу функции, при помощи параметра с именем *Result*, типа *TResult*. Для этого необходимо заполнить её поля данных и выставить флаг *Result→Save* в единицу. До сброса флага *Result→Save* в 0 работа с полями структуры запрещена, так как система производит необходимые внутренние операции по смене указателей на память.

Для контроля процесса работы DLL-библиотеки ядром системы предусмотрены флаги *Pause* и *Terminated*. При переданном флаге *Pause* не равном нолю, разработчик должен обеспечить паузу в работе функции *Solve* путём зависания в бесконечном цикле, используя команду «*Sleep(10)*». При флаге *Pause*, равном нолю, решение должно быть продолжено. При переданном флаге *Terminated* не равном нолю, разработчик должен прекратить решение задачи и выйти из функции *Solve* без возврата каких-либо результатов. Отсутствие обработки данных команд приведёт к распознанию DLL-библиотеки как некорректной и её удалению.

### 1.3.4 Обработка результатов

Сохранённые результаты проверяются и обрабатываются функцией *Result*. В случае завершения решения функция должна вернуть 1, иначе 0. При наличии результата, он, в соответствии со своим диапазоном, должен быть сохранён в файле, путь к которому передаётся в параметре с именем *FileResult*. При отсутствии исходного файла параметр с именем *FileSource* будет равен значению *NULL*.

В случае неуспешного решения или обработки всего диапазона [*Start*; *End*] система автоматически завершит задачу.

## 1.4 Диапазон

Диапазон — составляющая часть программы, которую пользователь может использовать в своих библиотеках, для полного перебора комбинаций символов различной длины, необходимых для решения задачи. Диапазон гарантирует, что при последовательном переборе будет получен упорядоченный набор всех возможных комбинаций символов заданного промежутка длин.

Диапазон состоит из 2 частей:

- *TDistance* — структура, хранящая текущее состояние диапазона, а также границы задачи для конкретного устройства;
- набор функций, предоставляющих возможность управления диапазоном.

### 1.4.1 Работа с диапазоном

Для управления состоянием диапазона пользователю доступны следующие функции [2]:

- *CreateBufferKey* — функция, выделяющая в памяти место (буфер) для выдачи комбинации символов;
- *DestroyBufferKey* — функция, освобождающая память (буфер) для выдачи комбинации символов;

- *GetKey* — функция для получения текущей комбинации символов;
- *GetNextKey* — функция для получения следующей комбинации символов;
- *Inc* — функция инкремента или увеличения состояния диапазона на заданное число шагов. В качестве аргумента может принимать объект типа *TStep* или любой целый тип данных. При вызове функции без указания шага состояние будет приведено к следующей комбинации;
- *IsEnd* — функция, возвращающая 1, если состояние диапазона установлено на следующее за последним состоянием, иначе 0.

Для того чтобы начать использовать диапазон, необходимо выделить место в памяти с помощью функции *CreateBufferKey*. Функция вернёт указатель на область памяти, которая в последующем будет заполнена набором символов.

Чтобы получить текущую комбинацию, необходимо вызвать функцию *GetKey* и передать в неё указатель, полученный из функции *CreateBufferKey*. При этом память по указанному адресу будет заполнена набором символов, а функция вернёт текущую длину комбинации.

Для последующего перебора достаточно использовать функцию *GetNextKey*, которая, как и функция *GetKey*, заполнит буфер следующим набором символов и вернёт его длину.

Если необходимо сделать шаг на несколько состояний вперёд, нужно вызвать функцию *Inc*, передав в неё значение шага. Если значение передано не будет, функция переведёт диапазон в следующее состояние.

По окончанию работы с диапазоном необходимо освободить выделенную память функцией *DestroyBufferKey*, передав в неё указатель, полученный из *CreateBufferKey*.

При использовании функций, меняющих состояние диапазона, в них, дополнительно, необходимо передавать ссылку на диапазон.

Формат всех описанных выше функций доступен на сайте проекта RuSolver после регистрации в качестве пользователя [3].

### 1.4.2 Структура диапазона

Структура диапазона состоит из данных, позволяющих получить упорядоченную последовательность комбинаций символов в пределах установленного диапазона.

Структура диапазона и его функции зависят от выбранного шаблона. Для константы *JClass*, равной числу 3, структура типа *TDistance* имеет следующие поля [2]:

- *Alphabet* — набор символов, определённых пользователем для использования в переборе. Количество символов ограничено 256 элементами;
- *AlphabetSize* — размер алфавита;
- *Password* — массив, представляющий комбинацию символов в виде последовательности индексов символов из алфавита;
- *MaxLen*, *MinLen*, *CurLen* — максимальная, минимальная и текущая длинна комбинации. *CurLen*, в отличие от других, меняется в процессе перебора. Минимальная длина диапазона может быть не менее одного символа, максимальная — не более 32 символов;
- *Start*, *End* — порядковые номера начальной и конечной комбинации для данного сегмента относительно общего набора комбинаций. Значение поля *Start* меняется во время перебора и обозначает текущий порядковый номер комбинации.

# ЛИТЕРАТУРА


1) М.В. Васильева, П.Е. Захаров, И.К. Сирдитов, П.А. Попов, М.С. Еремеева. Учебное пособие. Параллельное программирование на основе библиотек
2) Лаврентьев Д.Н. Спецификация вычислительной системы RuSolver.
3) Сайт проекта RuSolver, URL: http://rusolver.ru
4) Теодам А.А. Руководство разработчика RuSolver
5) Воеводин В. В. Параллельные вычисления. – Спб. БХВ-Петербург, 2002. — 608 с.