УДК 004.42

А.Н. Егоров, В.А. Кузнецов, В.Е. Марлей, И.А. Назаргулов

МОДЕЛЬ РАСПАРАЛЛЕЛИВАНИЯ ВЫЧИСЛЕНИЙ ДЛЯ ПОВЫШЕНИЯ ЭФФЕКТИВНОСТИ ВОССТАНОВЛЕНИЯ ДОСТУПА К ДАННЫМ В КОРПОРАТИВНЫХ СЕТЯХ

Егоров А.Н., Кузнецов В.А., Марлей В.Е., Назаргулов И.А. **Модель распараллеливания вычислений для повышения эффективности восстановления доступа к данным в корпоративных сетях.**

Аннотация. В статье рассмотрены модель и особенности реализации программной оболочки распараллеливания вычислений. Приведены результаты сравнительной оценки производительности решения задачи восстановления доступа к данным на различных аппаратных средствах, используя для этих целей, как последовательный алгоритм вычислений, так и реализацию на основе программной оболочки распараллеливания.

Ключевые слова: параллельные алгоритмы, технология CUDA, восстановление доступа к данным, восстановление ключа.

Yegorov A.N., Kuznetsov V.A., Marley V.Y., Nazargulov I.A. **The Model of Parallel Computing to Effectiveness Improvement of the Restoration of Access to Data in Corporate Networks.**

Abstract. In the article, the model and implementation feature of the program shell of parallelizing are considered. Results of the comparative performance evaluation of solving the task of access restoring to data on various hardware, using for these purposes, both a sequential algorithm of computing, and implementation on the basis of the program shell of parallelizing are given.

Keywords: parallel algorithms, CUDA technology, restoration of access to data, key recovery.

1. Введение. В условиях функционирования корпоративных сетей, в которых для передачи данных, как правило, используются защищенные каналы, предполагающие их передачу в зашифрованном виде, возникают сбои, связанные с потерей или искажением секретных ключей. В такой ситуации актуальной становится задача автоматизации восстановления этих ключей с целью обеспечения доступа к зашифрованным данным, что позволит в значительной степени снизить уровень влияния человека на восстановление работоспособности системы.

Для решения этой проблемы, в основном, применяют подходы [1, 2], предполагающие резервное копирование ключей при их генерации. Копии помещаются в хранилища, из которых в случае необходимости ключ можно восстановить. Политикой безопасности определяется формат таких копий: ключ хранится целиком в исходном или зашифрованном виде, либо он разделяется на части, которые помещаются в отдельные хранилища [3], и, наконец, может сохраняться ряд параметров, связанных с ключом [2, 4].

Хранилища реализуются как программными, так и аппаратными средствами [4], которые, помимо безопасного хранения ключа, могут поддерживать функции шифрования, дешифрования, генерации ключевой пары, а также хеш-функции, что в свою очередь значительно снижает нагрузку на программную часть криптографической системы.

Достоинством такого подхода является скорость восстановления ключа. Однако, хранение копий ключей является дополнительной уязвимостью системы и требует обеспечения надежной защиты хранилищ.

Для ее реализации могут применяться гибридные и (или) альтернативные методы. Например, в [2] резервирование ключа осуществляется в два этапа. Сначала вычисляются параметры ключа, а затем генерируется множество подобных данных того же формата, содержащих произвольные значения параметров, из которых невозможно восстановить ключ. В результате создается так называемый шум, маскирующий исходные параметры. Такой подход ведет к существенному увеличению времени восстановления ключа. Для решения этой проблемы применяют распределенную вычислительную систему с большим (порядка нескольких тысяч) количеством узлов [2].

Восстановление ключей может осуществляться автоматически либо с помощью одного или нескольких специальных агентов [5]. Одной из последних разработок в этой сфере является система HADM-KRS [6], которая реализует децентрализованный подход к организации взаимодействия клиента и агентов. Это позволяет отказаться от использования единого центра восстановления ключа, что снижает затраты на обслуживание и повышает доступность системы. Однако, использование дополнительного звена приводит к увеличению временных затрат на реализацию процесса восстановления.

В России был предложен один из эффективных методов надежного хранения и восстановления секретного ключа, основанный на применении нейросетевых преобразователей «Биометрия – код доступа» [7]. Подобные системы относятся к классу биометрических криптографических систем с генерацией ключа, особенность которых заключается в том, что ключ извлекается из биометрических данных пользователя и не хранится в базе данных [8]. Это исключает необходимость использования резервного копирования. Реализация подобных систем подразумевает использование специального и, зачастую, дорогостоящего оборудования, что снижает их привлекательность. Дополнительная проблема рассматриваемого метода возникает при долгосрочном хранении данных. Поскольку расшифровать их может только обладатель биометрических

параметров, то в случае, например, его увольнения, требуется предварительная перешифровка необходимых данных другим ключом.

В статье рассматривается реализация, при которой для восстановления ключа используется исходный и зашифрованный текст. Несомненно, такой подход требует существенных временных затрат на вычисления. Это связано с высокой степенью защиты современных криптографических систем.

Для сокращения времени обработки больших объемов информации применяют параллельные алгоритмы, повышение эффективности которых достигается увеличением числа одновременно работающих устройств (цифровой сигнальный процессор, центральный процессор, графический процессор, программируемая логическая интегральная схема и т.д.), используемых при распараллеливании.

Цель работы заключается в построении математической модели распараллеливания вычислений и ее интеграции в универсальную программную оболочку распараллеливания (ПОР) [9].

Рассмотрим механизмы, реализованные в ПОР, и эффективность ее применения для решения задачи восстановления доступа к данным, используя полный перебор возможных ключей.

2. Модель ПОР. Основное назначение программной реализации ПОР – организация эффективного исполнения прикладного решения пользователя. Для этого ПОР использует вычислительные мощности устройств компьютера.

ПОР представляет собой совокупность взаимодействующих модулей [10]. Связующим звеном этих модулей является ядро программной оболочки (ЯПО). ЯПО включает в себя базовую функциональность ПОР и имеет самую высокую независимость от других модулей. Вся основная логика работы ПОР заложена во взаимодействии ЯПО с модулями устройств. Модули устройств (МУ) выполняют основную работу вычислений ПОР – решение пользовательских задач (ПЗ). Предполагается также, что МУ имеют одно из двух состояний: busy (устройство занято) и free (устройство свободно).

При формировании требований к параллельным алгоритмам наиболее важным является определение последовательности выполнения определенных событий для каждого вычисления программы [11]. Это позволяет для описания модели ПОР использовать язык темпоральной логики линейного времени (LTL), в котором явно учтен феномен времени. Каждое событие будет характеризоваться булевой переменной, которая принимает значение «Истина» тогда и только тогда, когда наступает событие.

В LTL помимо булевых логических связок для описания причинно-следственной зависимости событий во времени используются темпоральные операторы:

- $X\varphi$ «в следующий момент времени», указывает на то, что φ выполняется на следующем состоянии пути;
- $F\varphi$ «когда-то в будущем», указывает на то, что φ будет соблюдаться в каком-то последующем состоянии пути;
- $G\varphi$ «всегда», указывает на то, что φ выполняется в каждом состоянии пути;
- $\varphi U\psi$ «до тех пор пока», указывает на то, что φ выполняется до тех пор, пока не станет верно ψ;
- $\varphi R\psi$ «высвободить, открепить», указывает на то, что ψ может перестать быть верным только после того, как станет верно φ.

Для описания модели введем следующие атомарные высказывания, соответствующие основным событиям вычислений программы:

- try_i – i-й пользователь собирается отправить задачу;
- add_i – i-я ПЗ добавляется в очередь задач;
- del_i – i-я ПЗ удаляется из очереди задач;
- $send_i$ – i-я ПЗ отправляется на решение МУ;
- $create_i$ – i-я подзадача формируется из ПЗ;
- $solve_i$ – i-й МУ решил подзадачу;
- $recv_i$ – i-й результат подзадачи МУ отправляется ЯПО;
- res_i – формируется i-й результат ПЗ;
- free – МУ свободен;
- busy – МУ занят;
- end – получен конечный результат ПЗ;
- empty – очередь задач пуста.

Рассмотрим математическую модель программной оболочки, построенную на основе языка LTL:

1. всякий раз, когда пользователь собирается запустить ПЗ, ЯПО добавляет себе ПЗ в очередь задач:

$$G\left(try_i \rightarrow add_j\right); \tag{1}$$

2. пока очередь задач не окажется пустой (empty), ЯПО будет отправлять задачи на решение свободным МУ (free):

$$G\left(\left(free \rightarrow send_j\right)\ U\ emplty\right); \tag{2}$$

3. всякий раз, когда ЯПО отправляет задачу МУ, формируется подзадача для этого модуля:

$$G\left(send_i \wedge\ X\ create_i\right); \tag{3}$$

4. всякий раз, когда у МУ формируется подзадача, он становится занятым (busy):

$$G\left(\mathrm{create}_j \rightarrow \mathrm{busy}\right); \tag{4}$$

5. всякий раз, когда МУ становится занятым, должна когда-нибудь решиться подзадача:

$$G\left(\mathrm{busy} \rightarrow \ F\,\mathrm{solve}_i\right); \tag{5}$$

6. после решения подзадачи МУ возвращает результат ЯПО, и становиться свободным (free):

$$G\left(\mathrm{solve}_i \rightarrow X\left(\mathrm{recv}_j \wedge \mathrm{free}\right)\right); \tag{6}$$

7. всякий раз, когда ЯПО получает результат решения подзадачи от МУ, ЯПО формирует результат соответствующей ПЗ:

$$G\left(\mathrm{recv}_i \rightarrow \ X\ \mathrm{res}_j\right); \tag{7}$$

8. всякий раз, когда ЯПО формирует результат решения ПЗ, оболочка проверяет, получен или нет конечный результат. Если этот результат получен, то ПЗ удаляется и очереди:

$$G\left(\left(\mathrm{res}_i \wedge \mathrm{end}\right) \wedge \mathrm{del}_j\right). \tag{8}$$

Полученные зависимости темпоральной логики позволяют представить псевдокод основных процессов модели ПОР (листинги 1, 2). Для каждой операции псевдокода ставится счетчик. Следует заметить, что псевдокод МУ является инвариантным по отношению ко всем вычислительным устройствам.

```
        while (true)
        {
1.1.        if (try)
1.2.            add();
1.3.        while (!empty && status == free)
1.4.            send();
1.5.        if (recv())
            {
1.6.            res();
1.7.            if (end)
1.8.                del();
            }
        }
```

Листинг 1. Псевдокод ЯПО модели ПОР

```
                while (true) {
2.1.                if (create()) {
2.2.                    status = busy;
2.3.                    while (!solve) {}
2.4.                    recv();
2.5.                    status = free;
                    }
                }
```

Листинг 2. Псевдокод МУ модели ПОР

В соответствии с построенным псевдокодом строится размеченная система переходов LTS (Labelled Transition System), где учитываются всевозможные состояния ПОР, основными из которых являются:

- счетчики выполнения команд;
- состояние решения задачи или подзадачи (решено/не решено);
- состояние модуля устройства (занят/свободен).

На рисунке 1 изображены схемы LTS каждого отдельного потока: единственного – для ЯПО и множества потоков МУ. В узлах через запятую показаны значения основных состояний ПОР.

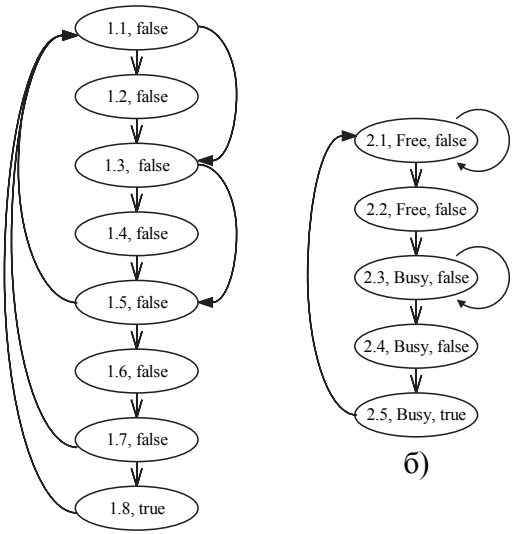

Рис. 1. LTS потоков ПОР: а) - ЯПО; б) - МУ

Поскольку программная реализация обеспечивает работу со многими процессами МУ, рассмотрим LTS–схему, демонстрирующую динамику взаимодействия двух процессов МУ и одного процесса ЯПО (рисунок 2).

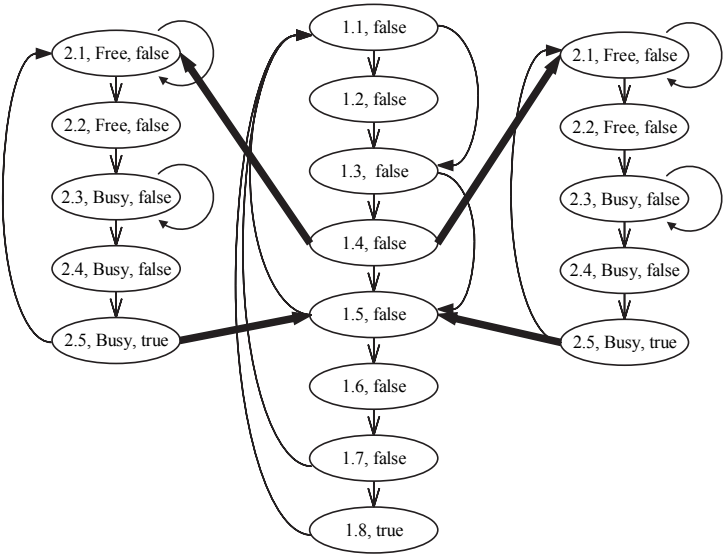

Рис. 2. LTS взаимодействия процесса ЯПО с двумя процессами МУ

ЯПО управляет работой процессов МУ, выполняемых параллельно. Взаимодействие МУ и ЯПО, осуществляемое во временные отметки отправки (блок состояния ЯПО – «1.4, false») и получения (блок состояния ЯПО – «1.5, false») результата решения подзадачи МУ, должно быть синхронизировано. На схеме синхронизация процессов показана жирными линиями.

3. Верификация модели ПОР. Обязательным этапов при разработке и реализации любых программных продуктов, включая в них параллельно исполняемые программы, является проверка корректности работы приложения. К параллельным программам могут успешно примяться следующие методы анализа корректности [12]:

1) тестирование;
2) формальная верификация: дедуктивная (доказательная); проверка на модели (model checking).

Э. Дейкстра доказал, что тестирование не позволяет разработчику доказательно удостовериться в корректности выполнения программы [13].

В общем случае формальная верификация представляет собой процесс доказательства с помощью формальных методов корректности или некорректности алгоритмов, программ и систем в соответствии с заданным описанием их свойств [12].

Задача дедуктивной верификации формулируется в виде доказательства теоремы (в системе математических доказательств), что предполагает огромную ручную работу, делая это метод малоприменимым на практике.

Верификацией на модели позволяет для заданной модели поведения системы с конечным, даже очень большим, числом состояний, проверить выполнимость некоторого логического требования (спецификации). Следует отметить, что спецификации обычно формулируют в терминах языка темпоральной логики. Это позволяет проверить не только условия на мгновенное состояние системы, но и историю его развития со временем.

Для верификации модели ПОР воспользуемся верификатором NuSMV, принимающим на вход модель, описанную на языке SMV. Указанный язык поддерживает задание спецификаций на языках CTL и LTL. Модель на языке SMV разделяется на модули, каждый из которых может содержать в себе набор переменных, правила переходов и требования [12].

В общем виде структура модели ПОР на языке SMV может быть представлена следующим образом (рисунок 3):

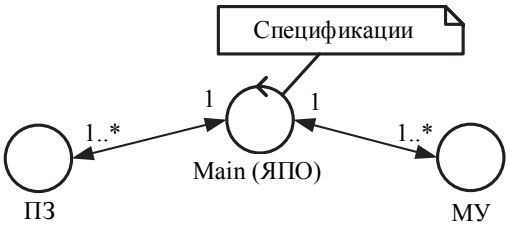

Рис. 3.Структура модели ПОР на языке SMV

Модуль main является обязательным для модели на языке SMV. При верификации модели ПОР данный модуль описывает алгоритм работы ЯПО согласно LTS потоков ПОР, представленных на рисунке 1.а). Main содержит спецификации, выдвигаемые к модели. Модуль МУ описывает логику работы программы в соответствии с LTS–схемой (рисунок 1.б). Для удобства описания модели ПОР на языке SMV состояния ПЗ и правила их переходов выделены в отдельном модуле.

В общем виде алгоритм проверки спецификаций, выдвигаемых к модели ПОР на языке SMV, описан схемой (рисунок 4). Схема логически разделена на 3 части, каждая из которых показывает действия, происходящие в соответствующем модуле.

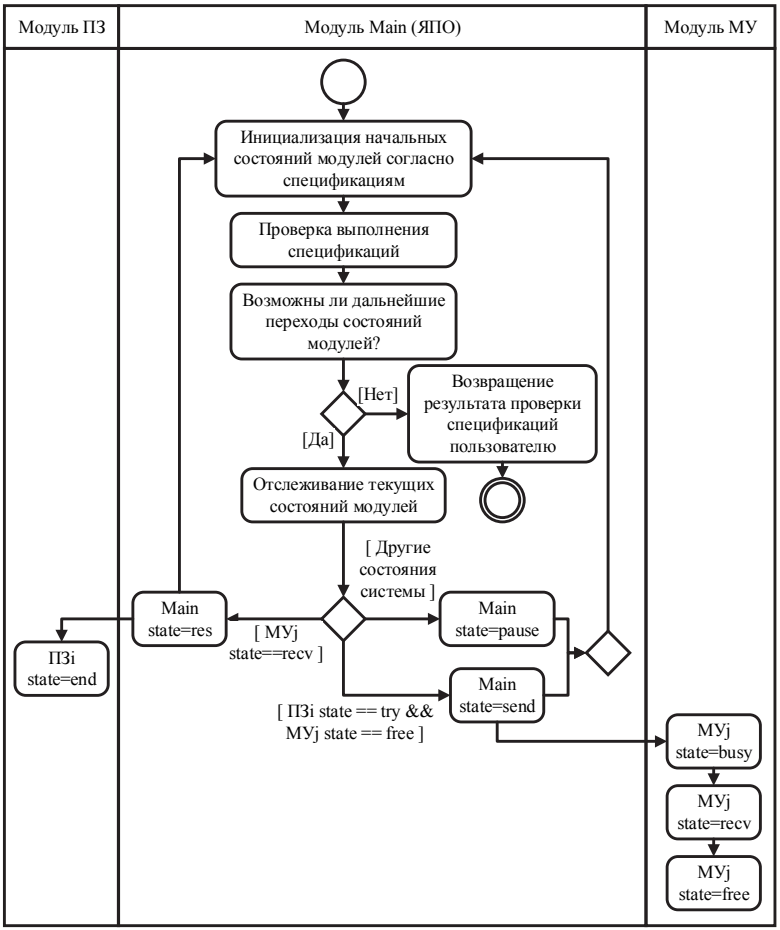

Рис 4. Обобщенная схема алгоритма проверка спецификаций выдвигаемых к модели ПОР на языке SMV

Функционал программной оболочки направлен на вычисления ПЗ, т.е., если пользователь отправляет задачу ПОР, то программная оболочка когда-нибудь должна вернуть результат решения этой

задачи. Данный факт необходимо проверить на модели. В терминах LTL требование выражается так:

$$G\left(try_i \rightarrow F\ end\right), \tag{9}$$

По результатам, приведенным на рисунке 5, видно, что модель ПОР соответствует требованию (9).

```
*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat
*** Copyright (c) 2003-2005, Niklas Een, Niklas Sorensson

NuSMV > process_model -i E:\por.txt
WARNING *** Processes are still supported, but deprecated.      ***
WARNING *** In the future processes may be no longer supported. ***

WARNING *** The model contains PROCESSes or ISAs. ***
WARNING *** The HRC hierarchy will not be usable. ***
The computation of reachable states has been completed.
The diameter of the FSM is 1.
-- specification  G (pz1.state = try ->  F pz1.state = end)  is true
-- specification  G (pz2.state = try ->  F pz2.state = end)  is true
-- specification  G (pz3.state = try ->  F pz3.state = end)  is true
NuSMV >
```

Рис. 5. Результат верификации ПОР верификатором NuSMV

4. Программная реализация. Программная реализация оболочки предназначена для распараллеливания процесса вычислений в прикладных решениях, используя для этих целей различные аппаратные средства. Каждому типу устройств соответствует свой обработчик. Поддержка прикладных решений реализуется в виде dll - библиотек, разработанных для каждого типа устройства, с которыми работает ПОР. Каждая библиотека должна содержать значения параметров, требуемых для настройки среды программной оболочки, таких как: размер массива бинарных данных задачи и результата ее решения; идентификатор задачи, и др. Так же необходимо реализовать ряд экспортных функций, обеспечивающих выполнение следующих операций [14]:

1. Score – оценка производительности устройства;
2. Parser – проверка, обработка и формирование данных ПЗ;
3. Cut – формирование данных подзадачи;
4. SimSim – решение подзадачи;
5. Result и ResultBuild – формирования общего решения ПЗ.

ПОР состоит из модулей, работу которых организует ЯПО [10]. Схема взаимодействия модулей программной реализации изображена на рисунке 6, на котором экспортные функции dll - библиотек представлены в виде отдельных модулей и выделены серым цветом. Сплошными линиями показаны связи между модулями, образующими

ЯПО, пунктирными динамика процесса решения ПЗ. Модули, взаимодействия которых обозначены жирными линиями, выполняются параллельно.

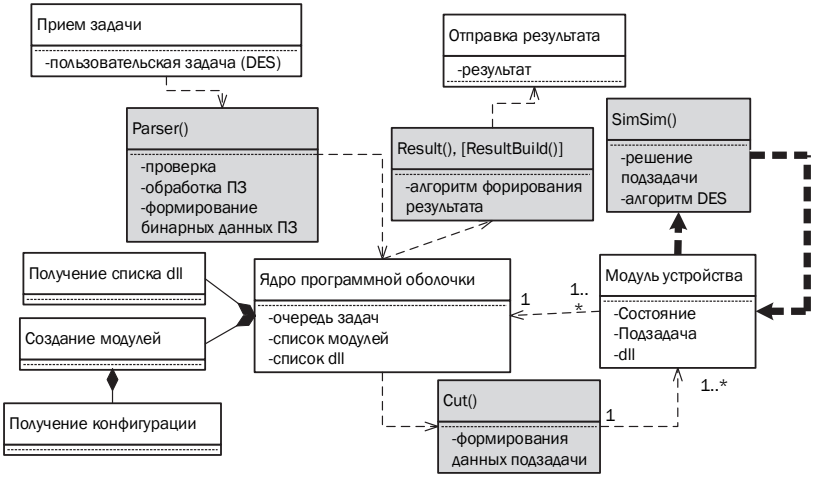

Рис. 6. Схема взаимодействия модулей ПОР

Работу ПОР опишем в виде пошагового алгоритма.

Этап 1. Создание модулей устройств (МУ), отвечающих за взаимодействие с конкретным аппаратным средством, на основе информации, сформированной в результате предварительного анализа вычислительных устройств компьютера, с которыми поддерживает работу ПОР.

Этап 2. Инициализация задачи посредством считывания и обработки данных конфигурационного файла, созданного пользователем, который содержит информацию о типе задачи и бинарные данные задачи, соответствующие выбранному типу.

Этап 3. Решение полученной задачи:

– шаг 1. С помощью функции «Score», вызванной из соответствующей dll - библиотеки, оценивается скорость решения текущей ПЗ на определенном устройстве;

– шаг 2. ЯПО отправляет задачу на решение множеству параллельно работающих МУ, при этом, в зависимости от оценки производительности устройства, модуль «Cute» формирует данные подзадачи для каждого МУ;

– шаг 3. В процессе решения подзадачи модуль «SimSim» осуществляет дополнительное распараллеливание вычислительного процесса, используя особенности аппаратного средства;

– шаг 4. После решения подзадачи МУ отправляет результат ЯПО.

Этап 4. Формирование общего результата решения задачи, на основе информации полученной от МУ, и отправка его пользователю.

При создании комплекта dll - библиотек для решения определенной задачи реализация функций «Score» и «SimSim», является уникальной по отношению к различным аппаратным средствам. В свою очередь, функции Parser, Cute, Result и ResultBuild, зачастую, являются инвариантными, однако в ряде случаев реализации отличаются. Например, в процессе распараллеливания вычислительного процесса функцией «SimSim» на конкретном устройстве могут понадобится дополнительные параметры, в связи с чем, формирование данных подзадачи функцией «Cute» будет другим.

Архитектура ПОР позволяет практически не ограниченно расширять список поддерживаемых устройств и задач. На текущий момент программная оболочка может использовать центральные процессоры и графические видеокарты, которые поддерживают технологию CUDA компании NVIDIA. CUDA – это платформа параллельных вычислений и модель программирования, позволяющая существенно увеличить вычислительную производительность за счет максимально эффективного использования ресурсов видеокарты (графических процессоров и памяти) компании NVIDIA.

В рамках задачи восстановления доступа к данным были созданы библиотеки для восстановления секретного ключа алгоритма шифрования DES. При реализации библиотек для отладки параллельных алгоритмов использовался NVIDIA Nsight, позволяющий оптимизировать производительность вычислений как для центрального, так и для графических процессоров. В общем виде реализованный по технологии CUDA алгоритм поиска ключа DES изображен на рисунке 7. Узлы, выполняемые в параллельных потоках, на схеме выделены серым цветом.

Вся совокупность потоков делится на блоки, каждый из которых работает со своим объемом встроенной памяти. Рациональная настройка, связанная с выбором количества таких блоков, в конечном итоге определяет количество потоков для распараллеливания.

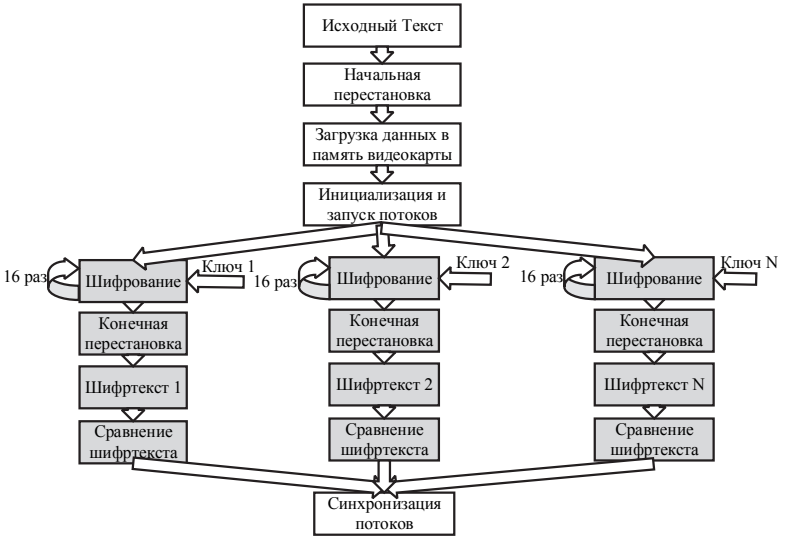

Рис. 7. Алгоритм поиска ключа DES на основе технологии CUDA

Количественная оценка производительности решения задачи восстановления ключа алгоритма шифрования DES в разработанной среде ПОР с применением технологий CUDA представлена в таблице 1. Значения таблицы, показывают эффективность применения ПОР для восстановления доступа к данным.

Таблица 1. Результаты оценки производительности решения задачи по восстановлению ключа алгоритма шифрования DES на конкретных вычислительных устройствах

Устройство	Режим решения		
	Последовательный (ключей/сек)	Параллельный на основе ПОР (ключей/сек)	Параллельный на основе ПОР с применением технологии CUDA (ключей/сек)
Процессор Intelcorei5	~160 000	~15 000 000	
Процессор Intelcorei7	~200 000	~20 000 000	
Видеокарта NVIDIAGTX 256			~156 000 000
Видеокарта NVIDIAGTX 590			~1 000 000 000

5. Заключение. Изложенный в статье подход к распараллеливанию вычислений при восстановлении секретного ключа позволяет поднять производительность процессоров до 10 раз. При использовании алгоритма на основе технологии CUDA выигрыш может достигать значения в 5000 раз по сравнению с последовательной реализацией. Учитывая тот факт, что в настоящее время персональный компьютер может иметь в комплекте до 7 видеокарт, эффективность их применения для параллельных вычислений не вызывает сомнений.

Дальнейшее развитие ПОР предполагает реализацию объединения компьютеров в единую вычислительную сеть и использование более эффективных алгоритмов для восстановлении секретных ключей, базирующихся на основе применения радужных таблиц для вычислений [15]. Это позволит еще существеннее поднять производительность.

Литература

1. *Полянская О.Ю., Горбатов В.С.* Инфраструктуры открытых ключей. Учебное пособие // М.: Интернет-Университет Информационных Технологий: БИНОМ. Лаборатория знаний. 2007. 367 с.
2. *Jajodia S., Litwin W., Schwarz Th.* Recoverable Encryption through a Noised Secret over a Large Cloud // Transactions on Large-Scale Data- and Knowledge-Centered Systems. 2013. vol. 9. pp. 42–64.
3. *Jajodia S., Litwin W., Schwarz Th.* LH*RE: A Scalable Distributed Data Structure with Recoverable Encryption // Proceedings third IEEE International Conference on Cloud Computing (CLOUD 2010). Miami. Fl. July 2010. pp. 354–361.
4. Способ и устройство для хранения и восстановления криптографического секретного ключа: пат. № 2279766. РФ. 2006. Бюл. № 19. 24 с.
5. *Barker E., Branstad D., Chokhani S., and Smid M.* A Framework for Designing Cryptographic Key Management Systems, Draft Special Publication 800–130. National Institute of Standards and Technology. 2010. 89 p.
6. *Kanyamee K., Sathitwiriyawong C.* High-availability decentralized cryptographic multi-agent key recovery // The International Arab Journal of Information Technology. 2014. vol. 11. no. 1. pp. 52–58.
7. *Гончаров С.М., Боршевников А.Е.* Построение нейросетевого преобразователя «Биометрия – код доступа» на основе параметров визуального вызванного потенциала электроэнцефалограммы // Доклады ТУСУР. 2014. Вып. 2(32). С. 51–55.
8. *Бардаев С.Э., Финько О.А.* Многофакторная биометрическая пороговая криптосистема // Известия ЮФУ. Технические науки. 2010. Вып 4. С. 148–155.
9. *Егоров А.Н., Кузнецов В.А., Назаргулов И.А.* Программная оболочка распараллеливания процесса вычисления прикладных решений. № 2014619266. РФ. 2014.
10. *Егоров А.Н., Кузнецов В.А., Назаргулов И.А.* Программная оболочка распараллеливания для восстановления доступа к данным // Региональная

информатика (РИ-2014) (г. Санкт-Петербург, 29-31 октября 2014 г.). Материалы конференции. СПб.: СПОИСУ. 2014. С. 71–72.
11. *Стальмаков В. А.* Параллельный генетический алгоритм для решения задачи составления расписания прохождения судов через шлюзованные системы и его верификация // Вестник ГУМРФ имени адмирала С. О. Макарова. СПб.: ГУМРФ имени адмирала С. О. Макарова. 2014. Вып. 1. С. 93–102.
12. *Чебатуркин А.А., Мазин М.А.* Методы верификации конечных автоматов, взаимодействующих по акторной модели // СПб.: ИТМО. 2010. 47 с.
13. *Dahl O.-J., Dijkstra E.W., Hoare C.A.R.* Structured Programming // Academic Press. 1972. 220 p.
14. *Егоров А.Н., Кузнецов В.А., Назаргулов И.А.* Параллельные алгоритмы для восстановления доступа к данным // Современные технологии и управление. Сборник научных трудов III Международной научно-практической конференции (р. п. Светлый Яр, 20–21 ноября 2014 г.). Светлый Яр: филиал ФГБОУ ВО МГУТУ имени К. Г. Разумовского (ПКУ) в р. п. Светлый Яр Волгоградской области. 2014. С. 39–42.
15. *Oechslin P.* Making a Faster Cryptoanalytic Time-Memory Trade-Off // Proceedings of the 23rd Annual International Advances in Cryptology. Santa Barbara. USA. 2003. vol. 2729. pp. 617–630.

References

1. Polyanskaya O.Yu., Gorbatov V.S. *Infrastruktury otkrytyh kljuchej* [Public key infrastructure]. M.: Internet-Universitet Informacionnyh Tehnologij: BINOM. Laboratorija znanij. 2007. 367 p. (In Russ.).
2. Jajodia S., Litwin W., Schwarz Th. Recoverable Encryption through a Noised Secret over a Large Cloud. Transactions on Large-Scale Data- and Knowledge-Centered Systems. 2013. vol 9. pp 42–64.
3. Jajodia S., Litwin W., Schwarz Th. LH*RE: A Scalable Distributed Data Structure with Recoverable Encryption. Proceedings third IEEE International Conference on Cloud Computing (CLOUD 2010). Miami. Fl. 2010. pp 354–361.
4. *Sposob i ustrojstvo dlja hranenija i vosstanovlenija kriptograficheskogo sekretnogo kljucha* [Method and device for the storage and recovery of a cryptographic secret key]: patent No. 2279766. Patent RF. 2006. bulletin 19. 24 p. (In Russ.).
5. Barker E., Branstad D., Chokhani S., and Smid M. A Framework for Designing Cryptographic Key Management Systems, Draft Special Publication 800–130. National Institute of Standards and Technology. 2010. 89 p.
6. Kanyamee K., Sathitwiriyawong C. High-availability decentralized cryptographic multi-agent key recovery. *The International Arab Journal of Information Technology*. 2014. vol. 11. no. 1. pp. 52–58.
7. Goncharov S.M., Borshevnikov A.E. [Construction of neural network transformer «Biometrics – access code» based on the parameters of the visual evoked potential electroencephalogram]. *Doklady TUSUR – Proceedings of TUSUR*. 2014. vol. 2. no. 32. pp. 51–55. (In Russ.).
8. Bardaev S.E., Finko O.A. [Multifactor biometric threshold cryptosystem]. *Izvestija JuFU. Tehnicheskie nauki – Izvestiya SFedU. Engineering sciences*. 2010. vol. 4. pp. 148–155. (In Russ.).
9. Yegorov A.N., Kuznetsov V.A., Nazargulov I.A. *Programmnaja obolochka rasparallelivanija processa vychislenija prikladnyh reshenij* [Program shell of parallelizing the computation of application solutions]. No. 2014619266. Patent RF. 2014. (In Russ.).

10. Yegorov A.N., Kuznetsov V.A., Nazargulov I.A. [Program shell of parallelizing for restoration of access to data]. *Regional'naja informatika (RI-2014). Materialy konferencii.* [Regional Informatics (RI-2014) Collected papers.]. St.Petersburg. 2014. pp. 71–72. (In Russ.).
11. Stal'makov. V.A. [Parallel genetic algorithm for solving scheduling passing through the gateways system and its verification]. *Vestnik GUMRF imeni admirala S.O. Makarova – Herald of Admiral Makarov SUMIS.* 2014. vol. 2. pp. 93–102. (In Russ.).
12. Chebaturkin A.A., Mazin M.A. *Metody verifikacii konechnyh avtomatov, vzaimodejstvujushhih po aktornoj modeli.* [Verification methods of finite automatons, interacting on actor model]. St.Petersburg: ITMO University Publ. 2014. 47 p. (In Russ.).
13. Dahl O.-J., Dijkstra E.W., Hoare C.A.R. Structured Programming. Academic Press. 1972. 220 p.
14. Yegorov A.N., Kuznetsov V.A., Nazargulov I.A. [Parallel algorithms for restoration of access to data]. *Sovremennye tehnologii i upravlenie. Sbornik trudov.* [Modern technology and management. Collected papers.]. Svetly Yar: Offices of MSUTM named after K.G. Razumovsky in Svetly Yar Publ. 2014. pp. 39–42. (In Russ.).
15. Oechslin P. Making a Faster Cryptoanalytic Time-Memory Trade-Off. Proceedings of the 23rd Annual International Advances in Cryptology. Santa Barbara. USA. 2003. vol. 2729. pp. 617–630.

Егоров Александр Николаевич — к-т техн. наук, доцент, профессор кафедры вычислительных систем и информатики, ФГБОУ ВО Государственный университет морского и речного флота имени адмирала С.О. Макарова. Область научных интересов: информационные системы и сети, технологии корпоративных сетей. Число научных публикаций — 52. eanspb@rambler.ru; ул. Двинская, д. 5/7, Санкт-Петербург, 198035; р.т.: +7(812)334-3874, Факс: +7(812)334-3874.

Yegorov Alexander Nikolayevich — Ph.D., associate professor, professor of computer system and informatics department, Admiral Makarov State University, of Maritime and Inland Shipping. Research interests: information systems and networks, technologies of enterprise networks. The number of publications — 52. eanspb@rambler.ru; 5/7, Dvinskaya str., St.Petersburg, 198035; office phone: +7(812)334-3874, Fax: +7(812)334-3874.

Кузнецов Виталий Александрович — аспирант, ФГБОУ ВО Государственный университет морского и речного флота имени адмирала С.О. Макарова. Область научных интересов: распределенные и параллельные системы, технология программирования, криптографические системы. Число научных публикаций — 2. kuznecov.v.spb@mail.ru; ул. Двинская, д. 5/7, Санкт-Петербург, 198035; р.т.: +7(904)616-59-03.

Kuznetsov Vitaliy Aleksandrovich — Ph.D. student, Admiral Makarov State University, of Maritime and Inland Shipping. Research interests: distributed and parallel systems, programming technology, cryptographic systems.. The number of publications — 2. kuznecov.v.spb@mail.ru; 5/7, Dvinskaya str., St.Petersburg, 198035; office phone: +7(904)616-59-03.

Марлей Владимир Евгеньевич — д-р техн. наук, профессор, заведующий кафедрой вычислительных систем и информатики, ФГБОУ ВО Государственный университет морского и речного флота имени адмирала С.О. Макарова. Область научных интересов: автоматизация моделирования и программирования, моделирование, моделирование исторического процесса, мониторинг состояния объектов, теоретическое

программирование, распределенные и параллельные системы. Число научных публикаций — 150. vmarley@rambler.ru, http://marley.spb.ru; ул. Двинская, д. 5/7, Санкт-Петербург, 198035; р.т.: +7(921)900-0006.

Marley Vladimir Yevgenyevich — Dr. Sci., professor, head of computer system and informatics department, Admiral Makarov State University, of Maritime and Inland Shipping. Research interests: automation of modeling and programming, simulation of the historical process, condition monitoring of objects, theoretical programming, distributed and parallel systems. The number of publications — 150. vmarley@rambler.ru, http://marley.spb.ru; 5/7, Dvinskaya str., St.Petersburg, 198035; office phone: +7(921)900-0006.

Назаргулов Ильшат Азатович — инженер-программист, компания «Като». Область научных интересов: информационные системы и сети, криптографические системы. Число научных публикаций — 2. nazargulov.i@gmail.com, https://kato.im/; ул. Нагорная, д. 22, кв. 33, дер. Первомайская, Мелеузовский район,республика Башкортостан, 453874; р.т.: 8(911)945-44-03.

Nazargulov Ilshat Azatovich — programming engineer, «Kato». Research interests: information systems and networks, cryptographic systems. The number of publications — 2. nazargulov.i@gmail.com, https://kato.im/; 22, Nagornaja str., apt. 33, Pervomajskija, Republic of Bashkortostan, 453874, Russia; office phone: 8(911)945-44-03.

РЕФЕРАТ

Егоров А.Н., Кузнецов В.А., Марлей В.Е., Назаргулов И.А. **Модель распараллеливания вычислений для повышения эффективности восстановления доступа к данным в корпоративных сетях.**

Рассмотренная в статье программная оболочка распараллеливания (ПОР) реализует универсальный подход к организации эффективного исполнения прикладных решений пользователя. В соответствии с этим утверждением практически любую задачу, требующую для повышения производительности использования распараллеливания вычислений, можно решить в этой программной среде, реализовав необходимые библиотеки. Каждая из этих библиотек должна содержать значения параметров, требуемых для настройки среды программной оболочки и функции, реализующие вычисление оценок производительности, необходимых для эффективного распределения нагрузки между вычислительными устройствами, инициализацию задачи и ее декомпозицию на подзадачи, решение подзадач и обработку результатов их решения.

Архитектура ПОР позволяет практически не ограниченно расширять список поддерживаемых устройств и задач. На текущий момент программная оболочка может использовать центральные процессоры и графические видеокарты, которые поддерживают технологию CUDA компании NVIDIA.

Изложенный в статье подход к распараллеливанию вычислений при восстановлении секретного ключа позволяет поднять производительность для процессоров до 10 раз. При использовании алгоритма на основе технологии CUDA выигрыш может достигать значения в 5000 раз по сравнению с последовательной реализацией. Учитывая тот факт, что в настоящее время персональный компьютер может иметь в комплекте до 7 видеокарт, эффективность их применения для параллельных вычислений не вызывает сомнений.

Дальнейшее развитие ПОР предполагает реализацию объединения компьютеров в единую вычислительную сеть и использование более эффективных алгоритмов для восстановлении секретных ключей, базирующихся на применении радужных таблиц для вычислений. Это позволит еще существеннее поднять производительность.

SUMMARY

Yegorov A.N., Kuznetsov V.A., Marley V.Y., Nazargulov I.A. **The Model of Parallel Computing to Effectiveness Improvement of the Restoration of Access to Data in Corporate Networks.**

The considered program shell of parallelizing implements a universal approach to organizing the effective execution of application solutions of a user. As consistent with this statement, almost any task a user is requiring to improve the performance of with the use of parallel computing, can be solved in this software environment by implementing the necessary libraries. Each of these libraries should contain the parameter values required to configure an environment of program shell and functions implementing power evaluation of hardware needed for effective load distribution between computing devices, tasks initialization and decomposition of the task into sub-task, sub-task solution and processing of the sub-task solutions.

Architecture program shell of parallelizing allows almost unlimited expanding the list of supported devices and tasks. At the moment the shell program can use central processors and graphic video cards which support the CUDA technology of the NVIDIA company.

The outlined approach to parallelization computing with restoring the secret key allows improving performance for processors by 10 times. When using algorithm on the basis of the CUDA technology the performance can improve by 5000 times in comparison with sequential implementation. Considering that now a personal computer can have in a set up to 7 video cards, efficiency of their use for parallel computing doesn't raise doubts.

Further development of program shell of parallelizing involves implementation of combining computers in a single computer network and use of more effective algorithms for restoration of the secret keys which are based on the use of rainbow tables for computation.