Федеральное государственное автономное учреждение
«Федеральный научный центр Научно-исследовательский институт системных исследований Национального исследовательского центра «Курчатовский институт»

(НИЦ «Курчатовский институт» - НИИСИ)

ТРУДЫ НИИСИ РАН

ТОМ 14 № 4

МАТЕМАТИЧЕСКОЕ И КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ СЛОЖНЫХ СИСТЕМ:

ТЕОРЕТИЧЕСКИЕ И ПРИКЛАДНЫЕ АСПЕКТЫ

МОСКВА
2024

Тематика номера:
Архитектура АСУ, проектирование и моделирование СБИС, микро- и наноэлектроника, высокопроизводительные вычисления, математические исследования

Журнал публикует оригинальные статьи по следующим областям исследований: математика, математическое и компьютерное моделирование, обработка изображений, визуализация, системный анализ, методы обработки сигналов, информационная безопасность, информационные технологии, высокопроизводительные вычисления, оптико-нейронные технологии, микро- и наноэлектроника, математические исследования и вопросы численного анализа, история науки и техники.

The topic of the issue:

Architecture of ACS, design and modeling of VLSI, micro- and nanoelectronics, high-performance computing, mathematical issues

The Journal publishes novel articles on the following research arias: mathematics, mathematical and computer modeling, image processing, visualization, system analysis, signal processing, information security, information technologies, high-performance computing, optical-neural technologies, micro- and nanoelectronics, mathematical researches and problems of numerical analysis, history of science and of technique.

Заведующий редакцией: В.Е.Текунов

Издатель: НИЦ «Курчатовский институт» - НИИСИ
117218, Москва, Нахимовский проспект 36, к. 1

СОДЕРЖАНИЕ

CONTENT

Организация средств защиты информации для ОСРВ Багет

Н.Д. Байков[1], А.Н. Годунов[2], В.Н. Родионов[3]

[1]НИЦ «Курчатовский институт» – НИИСИ, Москва, Россия, nknikita@niisi.ras.ru;

[2]НИЦ «Курчатовский институт» – НИИСИ, Москва, Россия, nkag@niisi.ras.ru;

[3]НИЦ «Курчатовский институт» – НИИСИ, Москва, Россия, rodionov@niisi.msk.ru

Аннотация. Рассмотрены методы, применяемые при реализации средств защиты информации для операционной системы реального времени (ОСРВ) Багет. Модель управления доступом основана на управлении доступом пользователей к командам. Описаны методы идентификации и аутентификации пользователей. Также описаны пользовательский интерфейс выполнения команд, программный интерфейс регистрации обработчиков пользовательских команд и средства конфигурирования ОСРВ.

Ключевые слова: ОСРВ Багет, средства защиты информации, идентификация, аутентификация, управление доступом.

1. Введение

Формально управляющие системы можно разделить на два типа:

- автоматические,
- автоматизированные.

Принадлежность к одному из этих типов определяется наличием или отсутствием в системе оператора – лица, осуществляющего эксплуатацию управляющей системы, включая обработку хранящейся в ней информации. В случае автоматических систем такой оператор отсутствует: управляющая система функционирует самостоятельно по заранее определенному алгоритму работы, описывающему логику выполнения прикладной задачи. Это исключает возможность появления в автоматической системе внутреннего нарушителя в лице оператора, пытающегося получить несанкционированный доступ к данным или нарушить доступность системы путем эксплуатации возможностей предоставляемого ему интерфейса. Если при эксплуатации автоматической управляющей системы дополнительно возможно обеспечить замкнутость среды функционирования (неизменность выполняемого ПО) или даже полностью изолировать систему от внешнего мира, это также позволяет гарантировать отсутствие внешнего нарушителя, воздействующего на систему из-за ее пределов. Благодаря этому во многих автоматических системах встраивание средств защиты информации либо оказывается избыточным в силу условий эксплуатации системы, либо требуется в ограниченном объеме для защиты от угроз при начальной установке/обновлении ПО, а также для защиты от ошибок на уровне программной реализации алгоритма прикладной задачи.

В автоматизированной системе выполнение рабочего процесса по определению предполагает участие в нем оператора, функции которого могут быть самыми разнообразными. Например, в задачи оператора могут входить мониторинг состояния системы, сбор и обработка полученной информации, а также ручное управление рабочими процессами. Также перечисленные функции могут быть распределены между несколькими операторами, осуществляющими взаимодействие с системой параллельно. Как следствие, возникают требования по разграничению доступов пользователей к функциям системы в зависимости от той роли, которую пользователь выполняет в автоматизируемом рабочем процессе. Настоящая работа посвящена описанию механизмов управления доступом, предоставляемых разработчикам автоматизированных систем на базе операционной системы реального времени (ОСРВ) Багет.

ОСРВ Багет – это семейство операционных систем реального времени, разработанное в НИЦ «Курчатовский институт» – НИИСИ для ЭВМ серии «Багет» [1]. В отличие от операционных систем общего назначения (например, MS Windows, операционных систем семейства Linux), ОСРВ Багет преимущественно используется во встраиваемых системах в автоматическом режиме функционирования. В последние годы, в связи с деятельностью, направленной на импортозамещение иностранного программного обеспечения на объектах критически важной инфраструктуры, наблюдается интерес к функциональным средствам ОСРВ, которые были бы способны обеспечить ее функционирование в автоматизированном режиме. В том числе, к ним относится комплекс средств защиты информации (КСЗИ), с помощью которого в прикладной

системе можно было бы организовать управление доступом. Общему описанию интерфейса КСЗИ ОСРВ Багет посвящен раздел 2 настоящей статьи. Поскольку проверка разрешения на выполнение действия возможна только при условии того, что пользователь, от имени которого запрашивается выполнение действия, известен, при построении модели управления доступом возникают задачи идентификации пользователя и его аутентификации (верификации того, что пользователь действительно является тем, кем он себя идентифицирует в системе). Данные вопросы рассмотрены в разделе 3. Раздел 4 посвящен описанию политики управления доступом в ОСРВ Багет. Дополнительно в разделе 5 рассмотрены особенности реализации подсистемы протоколирования.

2. Общее описание интерфейса КСЗИ в составе ОСРВ Багет

Перед началом рассмотрения особенностей реализации КСЗИ ОРСВ Багет необходимо отметить, что применение ОСРВ Багет в прикладных системах основано на использовании технологии кросс-разработки. Ее суть сводится к использованию двух ЭВМ: инструментальной и целевой. Инструментальная ЭВМ должна использоваться разработчиками для написания прикладного программного обеспечения (ППО). Как правило, в роли инструментальной ЭВМ выступает персональный компьютер под управлением операционной системы общего назначения (например, ОС Astra Linux). В случае ОСРВ Багет ППО разрабатывается на инструментальной ЭВМ в виде исходных текстов на языке программирования Си. Далее выполняется компиляция исходных текстов ППО и их компоновка с библиотеками из установленного на инструментальную ЭВМ дистрибутива ОСРВ. В результате получаются исполняемые файлы ядра ОСРВ и прикладных процессов. Ответственным за их формирование является интегратор, которому на этапе сборки дополнительно доступна настройка конфигурационных параметров ОСРВ, определяющих режим ее функционирования (в том числе параметров, отвечающих за режим функционирования встроенных в ОСРВ средств защиты информации). Подготовленные файлы далее загружаются на целевую ЭВМ, где взаимодействие с ППО осуществляют операторы системы (пользователи). Часть из них назначаются администраторами системы (безопасности): наделяются полномочиями по управлению доступами всех остальных пользователей. Таким образом, безопасность системы в общем случае зависит от трех групп лиц:

- администраторы;
- интегратор;
- разработчики ППО.

Для каждой из перечисленных групп лиц определен свой тип интерфейса, посредством которого они оказывают влияние на конечное поведение системы. В случае разработчиков ППО это программный интерфейс, состоящий из набора разрешенных для вызова в ППО функций языка Си. Для работы интегратора предназначен конфигуратор ОСРВ, при помощи которого на инструментальной ЭВМ настраиваются конфигурационные параметры ОСРВ (см. Рис. 1). Взаимодействие с ОСРВ администраторов и других пользователей на целевой ЭВМ основано на вводе команд в режиме командного интерпретатора и/или режиме веб-сервера.

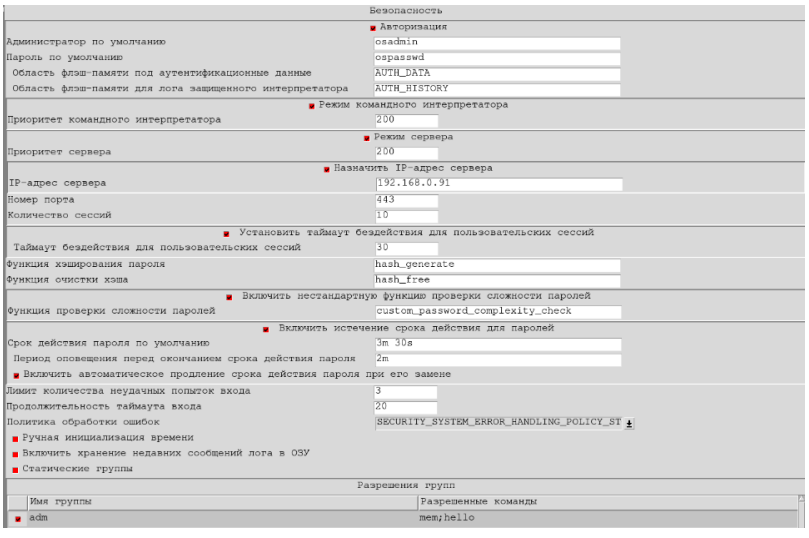

Рис. 1. Конфигурирование параметров безопасности ОСРВ на инструментальной ЭВМ

При конфигурировании ОСРВ интегратор должен указать, какие из этих двух способов ввода команд будут использоваться.

Если интегратором выбран способ ввода команд в режиме командного интерпретатора, при инициализации ОСРВ будет запущен специализированный поток выполнения, который будет заниматься обработкой поступающих в последовательный порт данных. При выборе этого режима дополнительно отключаются средства отладки ОСРВ, выполняющие чтение данных, поступающих в последовательный порт. В режиме интерпретатора команды обрабатываются последовательно. Пользователь вводит имя и пароль для входа в систему, после чего получает доступ к интерпретатору, в котором может последовательно вызывать команды. При наличии у пользователя достаточных прав будет вызван обработчик команды, и после его завершения управление будет возвращено пользователю для вызова следующей команды.

Если интегратором выбран способ отправки команд в режиме веб-сервера, при инициализации ОСРВ будет запущен поток выполнения, осуществляющий прием данных по сети. Обработка команд в этом случае осуществляется асинхронно. Веб-сервер выполняет роль диспетчера, который проверяет права доступа к командам и далее передает их потокам обработки команд. В начале пользователь посылает запрос на аутентификацию (по имени и паролю). В случае успеха в заголовках ответа будет передан идентификатор его сессии. Во всех последующих запросах на выполнение команд пользователю необходимо передавать назначенный ему идентификатор в заголовках запроса (куках).

Командный интерпретатор и веб-сервер могут быть включены одновременно.

Особенности, связанные с идентификаций и аутентификацией пользователей описаны в разделе 3. Правила управления доступом пользователей к командам описаны в разделе 4.

3. Идентификация и аутентификация пользователей

Идентификация и аутентификация пользователей с использованием КСЗИ ОСРВ Багет реализована при помощи имени и пароля. При каждом входе ОСРВ запрашивает у пользователя пароль и далее самостоятельно выполняет проверку этого пароля. В случае успеха пользователь считается аутентифицированным, т.е. подтвердившим, что он действительно является тем, кем он идентифицирует себя в системе. Ввиду отсутствия каких-либо средств делегирования, для верификации пароля пользователя требуется хранение в постоянной памяти целевой ЭВМ вспомогательных данных, связанных с учетными записями всех зарегистрированных в системе пользователей и достаточных для проведения процедуры верификации. Описанные данные называются аутентификационными данными. В простейшем случае в роли хранимых аутентификационных данных могли бы выступить сами пароли пользователей. Тогда для аутентификации пользователя было бы достаточно простого сравнения полученного от пользователя пароля с тем значением пароля, которое сохранено в памяти. Данный подход обладает недостатком: значения всех паролей пользователей в этом случае хранятся в памяти в открытом виде. Это автоматически означает, что при получении нарушителем доступа к чтению области памяти, в которой хранятся аутентификационные данные, все учетные записи мгновенно становятся скомпрометированными. По этой причине в большинстве систем, включая ОСРВ Багет, пароли напрямую не используются в качестве аутентификационных данных. Вместо этого используются производные от них хэши, получаемые в результате применения к исходному паролю пользователя специальной функции криптографического хэширования. Процедура аутентификации пользователя по паролю с использованием функции криптографического хэширования выглядит следующим образом:

- у пользователя запрашивается пароль;
- вычисляется хэш запрошенного пароля;
- вычисленное значение хэша сравнивается со значением хэша, хранящимся в постоянной памяти;
- в случае совпадения хэшей пользователь считается аутентифицированным.

Следует отметить, что формально в описанном алгоритме хэш-функция не обязана являться (и на практике, как правило, не является) инъективной. Т.е. алгоритм хэширования допускает существование коллизий – последовательностей символов, которым соответствует одинаковое значение хэша. Если одна из таких последовательностей используется в качестве пароля пользователя, все оставшиеся последовательности также могут быть использованы вместо пароля для его успешной аутентификации. С учетом этого для обеспечения требований безопасности необходимо, чтобы используемая при аутентификации хэш-функция обладала следующими свойствами:

- для одинаковых входных данных всегда должен получаться одинаковый результат вычисления;
- задача восстановления/подбора прообраза вычисленных значений должна быть избыточно трудоемкой – количество затрачиваемых на ее

решение ресурсов должно превышать потенциальную выгоду от получения несанкционированного доступа к системе в результате взлома пароля;

- задача поиска коллизий также должна быть избыточно трудоемкой.

Перечисленными свойствами обладает далеко не каждая хэш-функция. Разработка подобных функций является отдельным направлением исследований в информационной безопасности. Для использования на объектах критической инфраструктуры криптографическая хэш-функция должна пройти предварительную сертификацию на соответствие требованиям безопасности. По этой причине в ОСРВ Багет отсутствуют встроенные средства криптографического хэширования. Вместо этого предоставляется возможность подключения сертифицированных средств криптографического хэширования на этапе конфигурирования ОСРВ на инструментальной ЭВМ (см. Рис. 1). Интегратору необходимо указать имя функции, осуществляющей хэширование данных, и парную к ней функцию для освобождения ресурсов и очистки остаточного содержимого. Примером подходящей для заявленных целей функции может являться реализация алгоритма «Стрибог», описанного в межгосударственном криптографическом стандарте ГОСТ 34.11-2018 [2], разработанном на основе национального стандарта Российской Федерации ГОСТ Р 34.11.2012 [3].

Также следует отметить, что само по себе использование средств криптографического хэширования не гарантирует защиту паролей от их взлома. Уязвимость может содержаться в самом пароле. Многие пользователи используют в качестве паролей распространенные комбинации символов, что на порядки уменьшает для потенциального нарушителя область возможного поиска при переборе. В целях борьбы с этим в ОСРВ Багет предпринят ряд дополнительных мер безопасности:

- для защиты от взлома пароля путем перебора поддерживается возможность настройки таймаута при превышении лимита неудачных попыток входа в систему;

- неудачные попытки входа в систему протоколируются в журнале аудита, что позволяет администратору системы отследить попытки взлома путем перебора;

- предоставляется возможность задания в конфигурации ОСРВ пользовательской функции проверки сложности паролей, с помощью которой прикладная система может исключить использование слабых паролей, не отвечающих требованиям безопасности;

- поддерживается возможность ограничения сроков действия паролей пользователей по времени;

- хэши паролей хранятся и вычисляются с использованием «соли» (salt; см. ниже).

Понятие «соли» возникло как средство борьбы с использованием предварительно вычисленных таблиц хэшей от комбинаций символов, часто используемых пользователями различных информационных и управляющих систем в качестве паролей, т.к. с помощью таких таблиц нарушитель мог бы потенциально сократить время взлома учетных записей в случае утечки аутентификационных данных. Хотя утечка аутентификационных данных безусловно является угрозой безопасности системы, если удастся обнаружить ее до того, как нарушитель сможет вычленить из аутентификационных данных значения хэшей и восстановить их прообразы, это позволит избежать негативных последствий для системы. Будет достаточно выполнить замену паролей пользователей. Проблема заключается в том, что, если нарушителю известен используемый в системе алгоритм хэширования, он может заранее составить таблицу, состоящую из наиболее распространенных паролей, и предварительно вычислить все значения их хэшей. В том случае, если кем-то из пользователей системы используется пароль из таблицы, составленной нарушителем, время взлома такого пароля в случае утечки аутентификационных данных сократится практически до нуля, т.к. для этого будет достаточно выполнить поиск по таблице. Для предотвращения описанной угрозы необходимо сделать составление описанной таблицы нецелесообразным для нарушителя. Например, вынудить его увеличить объем таблицы до такой степени, чтобы на ее хранение и вычисление требовалось неоправданно большое количество ресурсов/времени. Для этого как минимум необходимо обязать всех пользователей системы использовать в качестве паролей достаточно сложные комбинации символов (не меньше установленной длины и задействующие как можно большее количество различных символов). Сделать это предлагается с помощью описанной выше настраиваемой функции проверки сложности паролей. Тем не менее, остается риск, связанный с тем, что нарушитель может приступить к составлению таблицы хэшей задолго до того, как в какой-либо из систем, использующих выбранный им алгоритм хэширования, возникнет утечка аутентификационных данных. Для того, чтобы сделать предварительное вычисление хэшей до утечки бессмысленным, в алгоритм аутентификации вводится соль. Под солью понимается произвольная последовательность символов, случайно формируемая в момент установки/изменения пароля пользователя.

Данная последовательность хранится в памяти в открытом виде и каждый раз добавляется к паролю (путем конкатенации) перед тем, как вычислить от него функцию криптографического хэширования (при этом хранящееся в аутентификационных данных значение хэша также должно быть получено с учетом соли). В таком случае нарушитель, который решил заранее вычислить таблицу хэшей распространенных паролей, столкнется с тем, что будет вынужден вычислять значения хэшей паролей с учетом всех возможных значений соли, поскольку на момент вычисления таблицы используемые в системе значения соли еще не известны нарушителю. Таким образом, увеличивая длину соли можно без усложнения требований к паролям пользователей добиться того, чтобы для хранения описанной таблицы у нарушителя оказалось недостаточно памяти.

4. Политика управления доступом

В случае ОСРВ объектами доступа являются команды. Команды делятся на встроенные и пользовательские. Все встроенные команды предопределены: они либо реализуют функции администрирования, либо относятся к общедоступным действиям. Например, к администрированию относится команда добавления нового пользователя `addUser`, а к общедоступным действиям относится команда выхода из системы в режиме командного интерпретатора `exit`. В отличие от встроенных, пользовательские команды предназначены для управления прикладной задачей. Список пользовательских команд варьируется от системы к системе. Добавление в систему пользовательских команд и назначение им функций-обработчиков выполняют разработчики ППО при помощи программного интерфейса. Для этого разработчику ППО необходимо реализовать алгоритм обработчика в виде функции языка Си и далее зарегистрировать обработчик, связав его с именем команды (например, при помощи предоставляемого ему для этого макроса `CUSTOM_CMD_DECL`). При этом ОСРВ обеспечивает передачу аргументов команды на вход функции-обработчику, а также предоставляет дополнительные функции для вывода результата выполнения команды: в последовательный порт в режиме командного интерпретатора и в тело ответа в режиме веб-сервера.

Политика управления доступом в ОСРВ Багет является ролевой. Роли пользователей реализуются через группы доступа. Описание каждой группы доступа состоит из уникального названия группы и списка разрешенных для ее членов команд. В список команд могут входить пользовательские команды и встроенные команды, для которых системой предусмотрена настройка доступа (например, возможность настройки предусмотрена для встроенной команды просмотра списка пользователей `users`). Для каждого пользователя в системе безопасности администратором задан атрибут, определяющий его принадлежность к группе доступа. Вызов команды доступен пользователю в следующих случаях:

- команда является встроенной и общедоступной;
- команда является пользовательской или встроенной с настраиваемым доступом, и пользователь состоит в группе, в списке разрешенных команд которой есть вызываемая команда;
- команда является встроенной командой администрирования, и пользователь является членом встроенной группы `adm` (зарезервирована для администраторов системы).

Согласно описанной политике управления доступом, решение о предоставлении аутентифицированному пользователю доступа к команде принимается только на основе проверки текущих значений атрибутов его учетной записи в системе. При этом результат вычисления не зависит от предыдущих действий этого и/или других пользователей системы (не считая изменений текущих значений атрибутов учетной записи пользователя в результате выполнения встроенных команд администратора). В частности, на уровне ОСРВ отсутствуют предопределенные ограничения на количество вызовов команды или доступность команды только в определенные интервалы времени и иные виды динамических ограничений. Модели доступа, обладающие описанными свойствами, принято относить к классу статических [4].

Дополнительной особенностью ОСРВ является наличие поддержки двух режимов работы с группами: статического и динамического. В динамическом режиме ответственным за создание групп и настройку доступных им команд являются администраторы системы на целевой ЭВМ. При выборе статического режима работы с группами действуют более жесткие ограничения на работу с группами. В этом случае администратору целевой ЭВМ доступно только включение пользователей в группы, а сам список групп и разрешенные им команды являются фиксированными и определяются интегратором на этапе конфигурирования ОСРВ на инструментальной ЭВМ. Предполагается, что режим статических групп может использоваться в системах с повышенными требованиями к разграничению информационных потоков для того, чтобы исключить потенциальное нарушения ролевой модели

прикладной системы в результате ошибочных действий администратора.

5. Подсистема протоколирования

Протоколирование событий безопасности, включая записи о действиях пользователей, является одним из общепринятых средств для выявления внутренних нарушителей. В ОСРВ Багет добавление записей в журнал событий работает в режиме кольцевого буфера. Его размер задается интегратором на этапе конфигурирования ОСРВ на инструментальной ЭВМ. Для хранения данных используется флэш-память целевой ЭВМ. При переходе к заполнению последнего из свободных секторов флэш-памяти система сигнализирует о приближающемся переполнении журнала. Команда очистки журнала доступна только администраторам, применяется к наиболее старым записям журнала и предотвращает его полное стирание. Таким образом, всегда известно, кто последним запрашивал очистку журнала (на случай, если внутренним нарушителем окажется администратор системы, который попытается скрыть следы своей деятельности).

Для целей аудита журнала предназначена встроенная команда `history`, позволяющая просматривать журнал и выполнять поиск по журналу с учетом примененных фильтров по времени и дате, номерам сессий пользователей, типам события, поисковой строке и другим дополнительным параметрам. Также ОСРВ поддерживает режим дублирования записей журнала в буфер оперативной памяти на случай, если требуется экспорт журнала аудита во внешние системы.

В журнале протоколируются все попытки вызова команд пользователями системы, результаты их выполнения, а также различные события безопасности. Например:

- начало и завершение каждой пользовательской сессии;
- неуспешные попытки аутентификации;
- предупреждения этапа инициализации системы безопасности ОСРВ (например, предупреждение о том, что не все пользовательские сессии были завершены при предыдущем запуске целевой ЭВМ, что может указывать на некорректное завершение работы).

Также программный интерфейс предусматривает наличие функции протоколирования, с помощью которой разработчики пользовательских команд могут добавлять собственные сообщения в журнал аудита.

6. Заключение

В работе рассмотрены особенности организации средств защиты информации, реализованных в составе ОСРВ Багет.

Публикация выполнена в рамках государственного задания по проведению фундаментальных исследований по теме «Создание и реализация доверенных систем искусственного интеллекта, основанных на новых математических и алгоритмических методах, моделях быстрых вычислений, реализуемых на отечественных вычислительных системах» (FNEF-2024-0001).

Security Features in RTOS Baget

N.D. Baykov, A.N. Godunov, V.N. Rodionov

Abstract. The methods used in implementing information security tools for the Baget real-time operating system (RTOS) are considered. The access control model is based on control of user access to commands. The methods for identifying and authenticating users are described. The user interface for executing commands, the software interface for registering user command handlers, and the RTOS configuration tools are also described.

Keywords: RTOS Baget, information security features, identification, authentication, access control

Литература

1. А.Н. Годунов, В.А. Солдатов. Операционные системы семейства Багет (сходства, отличия и перспективы) – «Программирование», Москва, 2014, № 5, 68–76.

2. ГОСТ 34.11-2018 «Информационная технология. Криптографическая защита информации. Функция хэширования». ФГУП «СТАНДАРТИНФОРМ». Москва. 2018.

3. ГОСТ Р 34.11-2012 «Информационная технология. Криптографическая защита информации. Функция хэширования». ФГУП «СТАНДАРТИНФОРМ». Москва. 2013.

4. V.C. Hu, R. Kuhn, D. Yaga. Verification and Test Methods for Access Control Policies/Models. Special Publication (NIST SP). National Institute of Standards and Technology. 2017. https://doi.org/10.6028/NIST.SP.800-192.

Библиотека поддержки протокола OPC UA для программируемых логических контроллеров семейства «Багет»

Т. К. Грингауз[1], Д. В. Яриков[2]

[1]НИЦ «Курчатовский институт» – НИИСИ, Москва, Россия, gring@niisi.ras.ru;

[2]НИЦ «Курчатовский институт» – НИИСИ, Москва, Россия, yarikov@ niisi.ras.ru

Аннотация. В среду отечественной операционной системы реального времени семейства Багет портирована свободно распространяемая библиотека open62541, реализующая протокол OPC UA. В статье приведен обзор базовых понятий OPC UA, перечислены функциональные возможности портированной версии библиотеки, описана модельная задача, приведен разбор ключевых фрагментов ее кода.

Ключевые слова: программируемый логический контроллер, протокол OPC UA, ОСРВ Багет

1. Введение

В ФГУ ФНЦ НИИСИ РАН ведется разработка линейки программируемых логических контроллеров на отечественной электронной компонентной базе (далее – «контроллеры семейства Багет»). Контроллеры предназначены для использования в автоматизированных системах управления технологическими процессами. В состав программного обеспечения контроллеров входит программа «Библиотека поддержки протокола OPC UA» (БПП OPC UA). Библиотека может использоваться приложениями, функционирующими в среде операционной системы реального времени ОСРВ Багет 2.7 [1], для предоставления или потребления данных посредством OPC UA[2].

Стандарт OPC UA (Unified Archtecture), известный как IEC 62541, определяет средства информационного моделирования данных и коммуникационный протокол для промышленных систем. Стандарт поддерживается некоммерческим Консорциумом OPC Foundation. Первая спецификация стандарта была представлена в 2006 г., а в 2012 г. стандарт был оформлен в рамках Единого реестра европейских стандартов [3]. К сильным сторонам стандарта, способствующим его широкому применению, относятся: аппаратно- и платформонезависимость, возможность построения на его основе сложных информационных моделей, сервис-ориентированная архитектура. Сервисы, специфицированные в OPC UA, определяют информацию, которой должны обмениваться приложения, но не конкретный способ коммуникации по сети, а также не конкретную реализацию посредством API.

Библиотека БПП OPC UA разработана на базе открытого исходного кода свободно распространяемого продукта open62541 [4]. Библиотека open62541 реализует стек бинарного протокола OPC UA, а также набор функций (SDK) для разработки на языке Си клиентских и серверных приложений OPC UA.

В статье приведен обзор базовых понятий OPC UA, перечислены функциональные возможности БПП OPC UA, описана модельная задача, приведен разбор ключевых фрагментов ее кода.

Изложение материала в разделе 2 основано на источниках [2],[5],[6].

2. Обзор OPC UA

2.1. Компоненты унифицированной архитектуры OPC UA

Унифицированная архитектура OPC UA включает в себя:

- метамодель для информационного моделирования, которая сочетает объектно-ориентированность с возможностью установления семантических отношений между объектами;
- асинхронный протокол (построенный на TCP, HTTP или SOAP), который определяет обмен сообщениями в рамках сессий, установленных над защищенными каналами связи;
- систему типов данных и схемы (двоичную и основанную на XML) их кодирования в сообщениях;
- набор из 37 стандартных сервисов для взаимодействия с серверными информационными моделями.

Совокупность механизмов кодирования (форматирования), защищенного канала и транспорта принято называть «стеком OPC UA».

Сервисы OPC UA - это интерфейс между серверами (поставщиками информационной модели) и клиентами (потребителями этой информационной модели). Сервисы используют стек OPC UA для обмена данными между клиентом и сервером.

2.2. Информационное моделирование в OPC UA

2.2.1. Информационная модель. Объектная модель. Адресное пространство

Информационная модель определяет, характеризует и связывает информационные ресурсы системы или набора систем, информацию о которых сервер предоставляет клиентам. Стандарт OPC UA специфицирует структуру объектно-ориентированных информационных моделей, предоставляемых сервером, и протокол, с помощью которого клиент может взаимодействовать с информационной моделью по сети.

В информационной модели OPC UA объект определяется как набор переменных, методов и ссылок на другие объекты. Объекты OPC UA типизированы. Тип определяет не только структуру объекта, но и его семантику. Для доступа к объектам и их компонентам клиент вызывает сервисы OPC UA. Сервисы обеспечивают чтение и запись переменных, вызов методов, создание экземпляров и удаление объектов, подписку на уведомления об изменениях и другие действия с объектами.

В OPC UA определен стандартный способ представления информационных моделей. Введено понятие «адресное пространство сервера» (далее – «адресное пространство»). Адресное пространство определено как «вся информация (collection of information), которую сервер делает видимой для своих клиентов» [2]. Информационная модель в адресном пространстве OPC UA представляется как граф, состоящий из узлов и ссылок между ними. Объект и его компоненты образуют связный подграф.

2.2.2. Модель узла

Узел определяется как набор атрибутов и ссылок. Атрибуты описывают узел, а ссылки определяют его связи (отношения) с другими узлами. В состав атрибутов входит обязательный атрибут NodeId - уникальный идентификатор узла внутри сервера.

2.2.3. Классы узлов

Узлы классифицированы. Определены 8 классов узлов:

-Variable (переменная);
-VariableType (тип переменной);
-Object (объект);
-ObjectType (тип объекта);
-ReferenceType (тип ссылки);
-DataType (тип данных);
-Method (метод);
-View (представление).

Каждый узел в адресном пространстве является экземпляром одного из восьми классов. Класс узла определяет состав набора его атрибутов и ссылок. Разные экземпляры одного класса имеют одинаковый набор атрибутов, различаться могут только их значения. Клиентам и серверам не разрешается определять дополнительные классы узлов или расширять список атрибутов для класса узла.

Класс узла идентифицируется атрибутом NodeClass:

```
typedef enum {
UA_NODECLASS_UNSPECIFIED = 0,
UA_NODECLASS_OBJECT = 1,
UA_NODECLASS_VARIABLE = 2,
UA_NODECLASS_METHOD = 4,
UA_NODECLASS_OBJECTTYPE = 8,
UA_NODECLASS_VARIABLETYPE = 16,
UA_NODECLASS_REFERENCETYPE = 32,
UA_NODECLASS_DATATYPE = 64,
UA_NODECLASS_VIEW = 128,
UA_NODECLASS_FORCE32BIT = 0x7fffffff
} UA_NodeClass;
```

Примечание – Здесь и далее для обозначения стандартных констант, атрибутов, типов данных и др. используются их имена из исходных текстов open62541.

2.2.4. Атрибуты

Атрибуты – это элементарные компоненты классов узлов. Они не видны напрямую в адресном пространстве, но их значения для конкретного узла могут быть запрошены или изменены клиентом посредством сервисов атрибутов (Atribute Service Set).

В спецификациях [2] определение атрибута включает в себя поля: идентификатор атрибута, имя, описание, типа данных и индикатор «обязательного/опциональный».

Каждому атрибуту присвоен числовой идентификатор UA_AttributeId. Всего в OPC UA определено 27 атрибутов с уникальными значениями идентификатора UA_AttributeId. В файле open62541/common.h определен тип данных UA_AttributeId (перечисление). Семантика большинства атрибутов будет описана при рассмотрении классов узлов в разделе 2.2.6 настоящей статьи.

Набор атрибутов, определенный для класса узла, не может расширяться клиентами или серверами. При создании узла в адресном пространстве необходимо указать значения обязательных

атрибутов его класса.

Следующие 7 атрибутов используются в определениях всех классов узлов (первые 4 обязательны, последние 3 – опциональны):

- NodeId (тип данных NodeId) - иденти-фикатор узла;
- NodeClass (тип данных NodeClass) – идентификатор класса узла);
- BrowseName (тип данных QualifiedName) – имя узла для сервера (не локализовано);
- DisplayName (тип данных LocalizedText) – имя узла для клиента (локализовано);
- Description (тип данных LocalizedText) – текстовое описание узла;
- WriteMask (тип данныхUint32) – специфицирует атрибуты, которые доступны для модификации клиентом;
- UserWriteMask (тип данныхUint32) – специфицирует атрибуты, которые доступны для модификации пользователем, подключенным в текущий момент.

Ниже приведены описания наборов атрибутов (из файла open62541/types.h), специфичных для отдельных классов узлов в БПП OPC UA:

```
typedef struct {
UA_UInt32 specifiedAttributes;
UA_LocalizedTextdisplayName;
UA_LocalizedText description;
UA_UInt32 writeMask;
UA_UInt32 userWriteMask;
UA_BooleanisAbstract;
UA_Boolean symmetric;
UA_LocalizedTextinverseName;
} UA_ReferenceTypeAttributes;

typedef struct {
UA_UInt32 specifiedAttributes;
UA_LocalizedTextdisplayName;
UA_LocalizedText description;
 UA_UInt32 writeMask;
UA_UInt32 userWriteMask;
UA_BooleanisAbstract;
} UA_ObjectTypeAttributes;

typedef struct {
UA_UInt32 specifiedAttributes;
UA_LocalizedTextdisplayName;
UA_LocalizedText description;
UA_UInt32 writeMask;
UA_UInt32 userWriteMask;
UA_ByteeventNotifier;
} UA_ObjectAttributes;

typedef struct {
UA_UInt32 specifiedAttributes;
UA_LocalizedTextdisplayName;
UA_LocalizedText description;
UA_UInt32 writeMask;
UA_UInt32 userWriteMask;
UA_Variant value;
UA_NodeIddataType;
UA_Int32 valueRank;
size_tarrayDimensionsSize;
UA_UInt32 *arrayDimensions;
UA_ByteaccessLevel;
UA_ByteuserAccessLevel;
UA_DoubleminimumSamplingInterval;
UA_Booleanhistorizing;
} UA_VariableAttributes;

typedef struct {
UA_UInt32 specifiedAttributes;
UA_LocalizedTextdisplayName;
UA_LocalizedText description;
UA_UInt32 writeMask;
UA_UInt32 userWriteMask;
UA_Variant value;
UA_NodeIddataType;
UA_Int32 valueRank;
size_t arrayDimensionsSize;
UA_UInt32 *arrayDimensions;
UA_BooleanisAbstract;
} UA_VariableTypeAttributes;

typedef struct {
UA_UInt32 specifiedAttributes;
UA_LocalizedTextdisplayName;
UA_LocalizedText description;
UA_UInt32 writeMask;
UA_UInt32 userWriteMask;
UA_Boolean executable;
UA_BooleanuserExecutable;
} UA_MethodAttributes;
```

2.2.5. Ссылки.

Ссылки используются для определения отношений узлов друг с другом. Тип ссылки является экземпляром класса узлов ReferenceType.

Ссылка представляет собой тройку вида <идентификатор узла-источника, идентификатор узла типа (ReferenceType), идентификатор узла-цели>.

Примером ссылки между узлами является ссылка типа «hasTypeDefinition» между узлами классов «переменная» и «тип переменной».

2.2.6. Определения классов узлов

2.2.6.1. Переменные (VariableNode)

Переменные используются для представления содержимого объекта, его реальных данных. Для этого класса специфичны следующие атрибуты:

- value (тип данных UA_Variant) – обязательный. Фактическое значение переменной. Тип данных значения определяется атрибутами DataType, ValueRank и ArrayDimensions;

- dataType (тип данных UA_NodeId) –обязательный. Идентификатор узла класса DataType;

- valueRank (тип данных UA_Int32) –обязательный. Указавает, является ли переменная массивом. Если является, то указывает число размерностей;

- arrayDimensions (тип данных UA_UInt32*) - опциональный. Для массива указывает число элементов по каждой размерности;

- accessLevel (тип данных UA_Byte) –обязательный. Битовая маска, указывающая, доступны ли на чтение и запись текущее значение и его история;

- userAccessLevel (тип данных UA_Byte) – обязательный.Битовая маска, указывающая, доступны ли на чтение и запись текущее значение и его история с учетом прав пользователя;

- minimumSamplingInterval (тип данных UA_Double) - опциональный. Минимальное время, за которое допустимо обновление значения переменной на сервере;

- historizing (тип данных UA_Boolean) – обязательный. Указывает, ведется ли сбор исторических данных в текущий момент.

Определены два вида переменных: свойства (Properties) и переменные данных (DataVariables). Свойства связаны с родительским узлом ссылкой типа "hasProperty" и не могут иметь дочерних узлов. Переменные данных могут иметь в качестве дочерних узлов свойства (тип ссылки "hasProperty") и другие переменные данных (тип ссылки "hasComponent").

2.2.6.2. Типы переменных (VariableTypeNode)

Узлы класса VariableNode создаются как дочерние по отношению к узлам класса VariableTypesNode (ссылка "hasTypeDefinition"). Родительские узлы определяют тип создаваемых переменных и накладывают ограничения на допустимые значения атрибутов dataType;valueRank, arrayDimensions для экземпляров переменных. Кроме того, при создании экземпляра переменной определенного типа дочернему узлу передается семантическая информация от родителя.

2.2.6.3. Объекты (ObjectNode)

Объекты используются для представления систем, компонентов систем, объектов реального мира и программных объектов. Объекты являются экземплярами типа объекта и могут содержать переменные, методы и другие объекты.

2.2.6.4. Типы объектов (ObjectTypeNode)

Класс узла ObjectType используется для представления типа объектов в адресном пространстве сервера. Узлы класса ObjectTypes аналогичны классам в объектно-ориентированных языках. Для класса узла ObjectType специфичен обязательный атрибут isAbstract (тип данных UA_Boolean). Он указывает, является ли тип абстрактным. Абстрактные типы не могут порождать экземпляров объектов.

Пример типа объекта – FolderType (тип «папка»). Объекты этого типа используются для представления адресного пространства в виде иерархии узлов. На рис. 1 изображен встроенный набор папок, задающий предопределенную структуру адресного пространства.

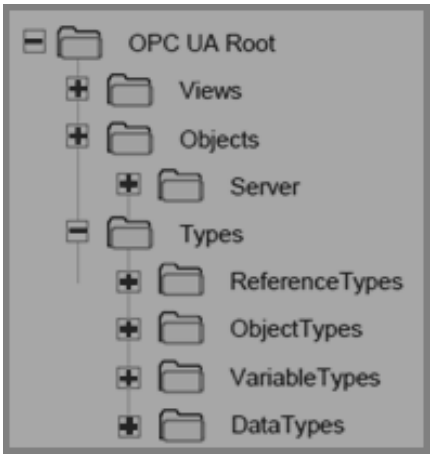

Рис.1 Стандартная структура адресного пространства ([2], Part 5)

2.2.6.5. Типы ссылок (ReferenceTypeNode)

Каждая ссылка между двумя узлами имеет тип, который определяет семантику отношения. Тип ссылки определяется узлом класса ReferenceType. Этот узел является родительским при создании ссылки.

Стандарт OPC UA определяет набор встроенных узлов ReferenceTypes. Их иерархия показана на рисунке 2 (стрелки указывают на отношение «hasSubType»). Полное описание семантики каждого типа ReferenceType содержится в части 3 спецификации OPC UA. Абстрактные типы ReferenceTypes не могут использоваться в реальных ссылках и применяются только для структурирования иерархии. Симметричные ссылки имеют одинаковый смысл для исходного и целевого узлов.

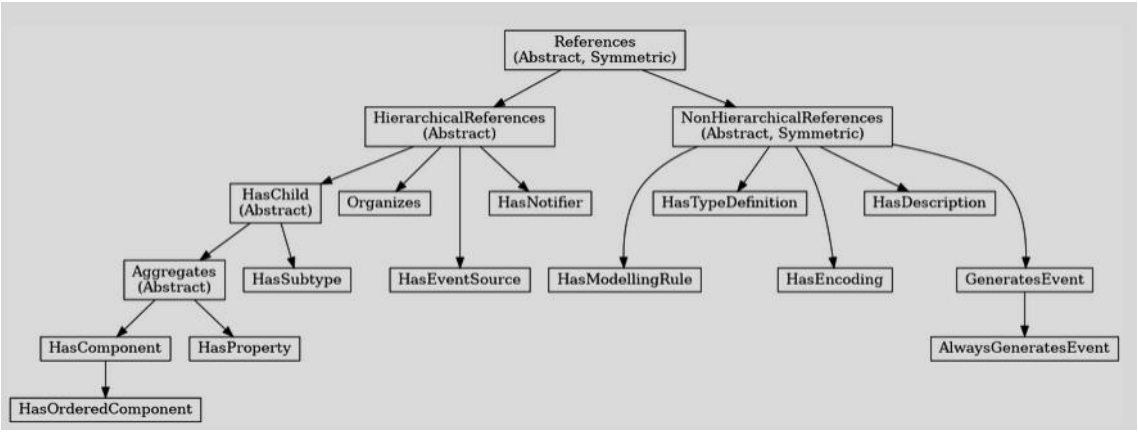

Рис. 2. Иерархия стандартных типов ссылки OPC UA (рисунок заимствован из [4])

2.2.6.6. Типы данных (DataTypeNode)

Узлы класса DataTypes представляют простые и структурированные типы данных. Типы DataTypes могут включать в себя массивы, но они всегда описывают структуру одного экземпляра. Абстрактные типы DataTypes (например, Number) не могут быть типом фактических значений. Они используются для ограничения значений набором дочерних DataTypes (например, UInt32).

2.2.6.7. Методы (MethodNode)

Методы определяют функции, которые вызываются с помощью сервиса Call. Узел MethodNode может иметь специальные свойства для определения входных и выходных аргументов. Эти свойства моделируются как дочерние узлы класса Variable со ссылкой hasProperty. Атрибут QualifiedName для этих переменных принимает значения (0, "InputArguments"), и (0, "OutputArguments") соответственно (стандартные имена, определенные в системном пространстве имен). Более подробно о пространствах имен написано в разделе 2.2.7 настоящей статьи.

Входные и выходные аргументы описываются как массив данных типа UA_Argument (стандартный тип OPC UA).

На один и тот же узел MethodNode могут ссылаться несколько объектов и типов объектов. В связи с этим запрос сервиса Call содержит идентификаторы NodeId и для вызываемого метода, и для объекта, предоставляющего контекст.

Следующие два обязательных атрибута с типом данных UA_Boolean специфичны для класса узлов MethodNode: executable, userExecutable. Они показывают, доступен ли метод для вызова в текущий момент.

2.2.6.8. Представления (ViewNode)

Каждое представление определяет подмножество узлов в адресном пространстве. Представления можно использовать при просмотре информационной модели, чтобы сосредоточиться только на интересующем пользователей подмножестве узлов и ссылок. Узлы класса ViewNodes можно создавать и взаимодействовать с ними, но их просмотр с помощью сервиса Browse в настоящее время не поддерживается в open62541.

2.2.7. Идентификация узлов в адресном пространстве. Тип данных UA_NodeId. Пространства имен (NameSpases)

Каждый узел в адресном пространстве имеет уникальный идентификатор в рамках сервера. Идентификаторы имеют стандартный тип данных UA_NodeId. Приведем определение этого типа из файла open62541/types.h:

```
enum UA_NodeIdType {
UA_NODEIDTYPE_NUMERIC   = 0, /* In the binary encoding, this can also become 1 or 2 (two-byte and four-byte  encoding of small numeric nodeids) */
    UA_NODEIDTYPE_STRING        = 3,
    UA_NODEIDTYPE_GUID          = 4,
    UA_NODEIDTYPE_BYTESTRING = 5
};
typedef struct {
    UA_UInt16 namespaceIndex;
    enum UA_NodeIdType identifierType;
    union {
        UA_UInt32    numeric;
        UA_String    string;
        UA_Guid      guid;
        UA_ByteString byteString;
```

} identifier;} UA_NodeId;

Поле namespaceIndex имеет следующий смысл. С каждым экземпляром сервера связано не менее двух пространств имен, из которых могут выбираться идентификаторы. URI этих пространств регистрируются в массиве NamespaceArray. В пространстве имен каждое имя уникально. Одинаковые имена из разных пространств должны относиться к разным узлам. Поле namespaceIndex указывает индекс пространства имен идентификатора в массиве NamespaceArray. Индекс 0 зарезервирован для предопределенного пространства имен от OPC Foundation (далее – «системное пространство имен». Индекс 1 определяет пространство имен сервера. Существуют и другие пространства имен, которые могут быть зарегистрированы в массиве.

В файле open62541/types.h определены static UA_INLINE - функции для упрощения создания идентификатора узла:

- static UA_INLINE UA_NodeId UA_NODEID_NUMERIC(UA_UInt16 nsIndex, UA_UInt32 identifier);
- static UA_INLINE UA_NodeId UA_NODEID_STRING(UA_UInt16 nsIndex, char *chars).

В файле open62541/ns0ids.h приведены идентификаторы всех предопределенных узлов адресного пространства.

2.3 Концепция коммуникационного протокола OPC UA

2.3.1 Транспортный уровень

Стандарт OPC UA определяет коммуникацию на основе транспортных механизмов TCP, HTTP и SOAP, определенную в стандарте. Программная реализация open62541 использует двоичный протоколе на основе TCP, поскольку он является наиболее распространенным транспортным уровнем для OPC UA.

TCP-соединение открывается для соответствующего имени хоста и порта. При открытии соединения устанавливаются основные настройки соединения, такие как максимальная длина сообщения.

2.3.2 Протокол «клиент/сервер»

В OPC UA определены два способа обмена данными между серверами и клиентами: «клиент/сервер» (Client/Server) и «Издатель/ Подписчик» (Pub/Sub). Мы ограничимся рассмотрением классической технологии «клиент/сервер», основанной на принципе «Запрос/Ответ». Клиенты отправляют запросы на сервер. Сервер отправляет ответы только на соответствующие запросы.

Протокол OPC UA допускает асинхронные ответы. Сервер не должен немедленно отвечать на запросы, и ответы могут отправляться в другом порядке. Такая необходимость возникает в случаях, когда требуется значимое время для обработки запроса (например, при вызове метода или при считывании данных с датчика с задержкой). Для подобных случаев стандарт OPC UA определяет в рамках клиент-серверного протокола технологию «подписки» (Subscrip-tions). Подписки реализуются с помощью специальных запросов, где ответ задерживается до публикации уведомления.

Примечание – Технологию подписки, поддерживаемую в рамках клиент-серверного протокола OPC UA, следует отличать от технологии PubSub, определенной в части 14 стандарта. PubSub – это расширение концепции подписки с целью интеграции коммуникации «многие ко многим» с OPC UA. PubSub не использует протокол клиент-сервер. Подписки реализуют индивидуальную связь клиента с сервером.

Клиент-серверное соединение для двоичного протокола OPC UA состоит из трех вложенных уровней: TCP-соединение с сохранением состояния, безопасный канал связи (Secure-Channel), сеанс связи (Session, далее – «сессия»).

2.3.3 Настройки безопасности. Конечные точки (endpoint)

Чтобы подключиться к серверу, клиенту необходимо знать сетевой адрес, протокол и настройки безопасности.

Информация, необходимая для установления соединения между клиентом и сервером, хранится в так называемой конечной точке сервера (endpoint). Сервер может предоставить несколько конечных точек. Конечная точка представляет собой набор следующих данных:

- URL-адрес конечной точки (протокол и сетевой адрес);
- политика безопасности (название набора алгоритмов безопасности и длина ключа шифрования);
- SecurityMode (уровень безопасности для обмена сообщениями);
- тип аутентификации пользователя.

Сервис GetEndpoints возвращает список доступных конечных точек. Этот сервис можно вызвать без открытия сессии и без шифрования сообщений. Это позволяет клиентам сначала обнаружить доступные конечные точки, а затем использовать соответствующую политику безопасности, которая может потребоваться для открытия сессии.

2.3.4 Безопасный канал связи (SecureChannel)

Безопасные каналы SecureChannels создаются над TCP-соединением. Канал SecureChannel устанавливается по запросу OpenSecure-Channel.

Безопасный канал характеризуется уровнем безопасности передачи сообщений (SecurityMode) и политикой безопасности (Security Policy).

Определены три допустимых уровня безопасности для канала SecureChannel: None, Sign, SignAndEncrypt.

При уровнях безопасности с подписью или шифрованием (Sign или SignAndEncrypt) сообщения шифруются с использованием асимметричного алгоритма шифрования (криптография с открытым ключом). В рамках сервиса OpenSecureChannel клиент и сервер устанавливают общий секрет по изначально незащищенному каналу. Для последующих сообщений общий секрет используется для симметричного шифрования, которое выполняется намного быстрее.

Различные политики безопасности, специфицированные в части 7 стандарта OPC UA, определяют алгоритмы для асимметричного и симметричного шифрования, длины ключей шифрования, хэш-функции для подписи сообщений и т. д. Примеры политик безопасности: None, Basic256Sha256.

2.3.5 Сессия (Session)

Сессия создается над каналом SecureChannel. Сессия гарантирует пользователям возможность аутентификации без отправки своих учетных данных в открытом виде.

Определены следующие механизмы аутентификации: анонимный вход, имя пользователя/пароль, сертификаты Kerberos и x509.

Для установления сессии используются два сервиса: CreateSession и ActivateSession. Сервис ActivateSession может использоваться для переключения существующей сессии на другой канал SecureChannel. Такое переключение позволяет повторно использовать существующую сессию при разрыве соединения.

2.3.6 Структура сообщения

Сообщение протокола OPC UA состоит из заголовка и тела.

Структура заголовка одинакова для всех сообщений и определяется стандартным типом данных OPC UA. Заголовок сообщения содержит базовую информацию, включая длину сообщения, идентификаторы сессии и канала SecureChannel, а также данные для связи запроса с соответствующим ответом. Специальное поле «Chunking» («разбиение на фрагменты») определяет способ разделения и повторной сборки сообщений, длина которых превышает максимальный размер сетевого пакета.

Структура тела сообщения зависит от того, какой сервис его передает, и от того, является ли оно запросом или ответом. Для каждого сервиса в системе типов OPC UA определены типы данных, соответствующие его запросу и ответу. Текст сообщения начинается с идентификатора типа данных, затем следует основная полезная нагрузка сообщения.

Примечание – Определения структур и идентификаторов типов данных, соответствующих запросам и ответам сервисов, содержатся в файле open62541/types_generated.h.

3 Общая характеристика БПП OPC UA

3.1 Стек OPC UA

Стек БПП OPC UA унаследован от библиотеки open62541 и обладает следующими характеристиками [4]:

- транспортные механизмы реализованы в соответствии с профилем UA-TCP UA-SCUA-Binary. Этот профиль определяет комбинацию сетевого протокола, протокола безопасности и кодирования сообщений, оптимизированную для низкого потребления ресурсов и высокой производительности. Он включает в себя сетевой протокол на основе TCP UA-TCP 1.0, бинарный протокол безопасности UA-SecureConversation 1.0 и бинарное кодирование сообщений UA-Binary 1.0;

- поддерживаются следующие виды шифрования сообщений: None, Basic128Rsa15, Basic256, Basic256Sha 256, Aes128Sha256-RsaOaep;

- поддерживаются следующие виды аутентификации: Anonymous, User Name Password, X509 Certificate.

3.2 SDK для разработки серверных приложений

В БПП OPC UA поддерживаются следующие возможности сервера:

- поддержка всех типов узлов OPC UA;

- контроль доступа для отдельных узлов;

- поддержка создания информационных моделей на стороне сервера на основе стандартных определений XML (наборов узлов);

- поддержка добавления и удаления узлов и ссылок во время выполнения;

- поддержка наследования и создания экземпляров типов объектов и переменных (пользовательский конструктор/деструктор, создание экземпляров дочерних узлов);

- поддержка подписок (Subscriptions)/

контролируемых элементов (MonitoredItems) (уведомления при изменении данных).

В дистрибутив БПП OPC UA входит пример сервера (server_ctt), созданный с использованием open62541 v1.0, реализованный в соответствии с профилем OPC Foundation «Micro Embedded Device Server» ([2]: Part 7: Profiles). Профиль поддерживает следующие возможности, специфицированные в стандарте OPC UA: связь клиент/сервер, подписки, вызовы методов и безопасность с политиками безопасности «Basic128Rsa15», «Basic256», «Basic256-Sha256», а также вызов методов и управление узлами.

3.3 SDK для разработки клиентских приложений

SDK для разработки клиентских приложений позволяет реализовывать следующие сервисы:

- DiscoveryServiceSet – группа сервисов обнаружения доступных серверов в составе: FindServers, GetEndpoints,RegisterServer;

- SecureChannelServiceSet – группа сервисов формирования безопасного канала в составе: OpenSecureChannel, CloseSecureChannel;

- SessionServiceSet – группа сервисов формирования сессии в составе: Session Service Set, CloseSession, .ActivateSession;

- Node Management Service Set – группа сервисов управления узлами в составе: AddNodes, AddReferences, DeleteNodes, DeleteReferences;

- ViewServiceSet – группа сервисов работы с представлениямивсоставе: BrowseNext, TranslateBrowsePathsToNodeIds, RegisterNodes, UnregisterNodes;

- AttributeServiceSet – группа сервисов управления атрибутами в составе: Read,Write;

- MethodServiceSet-группа сервисов вызова методов в составе: Call;

- MonitoredItemsServiceSet – группа сервисов для работы с отслеживаемыми объектами (позволяет клиенту создавать отслеживаемые объекты и привязывать их к переменным, атрибутам или событиям) в составе: Create-MonitoredItems, DeleteMonitoredItems, Modify-MonitoredItems, SetMonitoringMode, SetTrig-gering;

- Subscription Service Set – группа сервисов управления подписками в составе: CreateSubscription, ModifySubscription, Set-Publishing-Mode, Publish, Republish, DeleteSubscriptions, TransferSubscriptions.

Примечание – Выше перечислены только те сервисы из состава библиотеки open62541, для которых на текущий момент подтверждена работоспособность в составе БПП OPC UA.

4 Описание модельной задачи

Задача выполняется на программируемом логическом контроллере, содержащем в своем составе процессорный модуль и два модуля ввода-вывода: для дискретных и аналоговых сигналов соответственно. Модули ввода-вывода соединены с процессорным модулем шиной Modbus. Процессорный модуль функционирует под управлением операционной системы ОСРВ Багет 2.7.

На процессорном модуле функционирует сервер OPC UA, разработанный с помощью БПП OPC UA. В адресном пространстве сервера создано два объекта. Каждый объект содержит по 4 переменных. Значения переменных обновляются с частотой 100 мсек. Объект 1 (DI) принимает данные с модуля дискретных сигналов, объект 2 (AI) – с модуля аналоговых сигналов.

Данные объекта 1 получает OPC UA -клиент, функционирующий на инструментальной ЭВМ с микропроцессорной архитектурой x86 под управлением операционной системы семейства Linux. В качестве клиента OPC UA используется свободно распространяемая программа UAExpert с графическим интерфейсом (разработка фирмы UnifiedAutomation). Клиент получает данные с ПЛК путем синхронных запросов по технологии «клиент-сервер».

Данные объекта 2 получает OPC UA -клиент, разработанный на основе Библиотеки OPC UA, функционирующий локально на процессорном модуле ПЛК. Клиент взаимодействует с сервером OPC UA по протоколу TCP. Используется соединение по защищенному каналу с режимом подписи и шифрования. Самоподписанный сертификат и закрытый ключ шифрования сгенерированы посредством программы Open-SSL. Клиент получает данные по подписке.

Отображение адресного пространства сервера в окне программы UAExpert приведено на рис. 3.

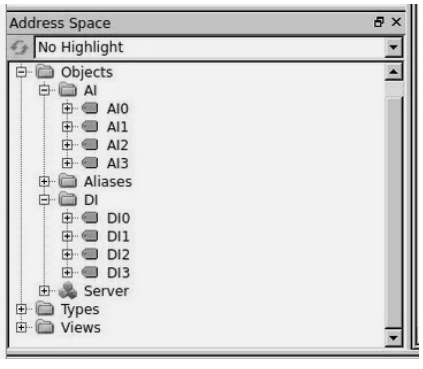

Рис.3 Отображение адресного пространства сервера в окне программы UAExpert.

Ключевые фрагменты кода, использованные при разработке сервера на основе БПП OPC UA, прокомментированы в разделе 8 настоящей статьи.

Примечание – Применение сервисов Subscriptions, MonitoredItems реализовано в клиентской части приложения и не комментируется в настоящей статье.

5 Ключевые фрагменты кода модельной задачи

5.1 Структура серверного приложения

Разработано серверное приложение на основе API БПП OPC UA, которое:

- конфигурирует и запускает сервер OPC UA;
- создает объекты и переменные адресного пространства;
- периодически обновляет в реальном времени значения переменных.

За основу был взят исходный текст примера examples/server_inheritance.c из дистрибутива БПП OPC UA.

Программа, реализующая серверное приложение, структурирована следующим образом.

Головная функция int main(void):

- создает и инициализирует конфигурацию сервера в соответствии с потребностями задачи;
- вызывает пользовательскую функцию createCustomInheritance, которая создает информационную модель задачи в адресном пространстве сервера;
- привязывает к конфигурации сервера пользовательскую Callback-функцию testCallback, которая устанавливает связь переменных с внешними источниками данных;
- запускает сервер;
- по завершении работы сервера освобождает ресурсы.

5.2 Конфигурирование сервера

Конфигурационные данные сервера OPC UA (далее – «конфигурация») используются при инициализации сервера. Структура конфигурации определена в заголовочном файле open62541/server.h: struct UA_ServerConfig {}.

Примеры полей структуры UA_ServerConfig:

UA_ApplicationDescription applicationDescription;

– описание приложения;

UA_ByteString serverCertificate;

– сертификат сервера,

const UA_DataTypeArray *customDataTypes;

– массив пользовательских типов данных;

size_t securityPoliciesSize;

UA_SecurityPolicy* securityPolicies;

– массив URI поддерживаемых политик безопасности;

size_t endpointsSize;

UA_EndpointDescription *endpoints;

– список конечных точек;

UA_UInt16 maxSecureChannels;

UA_UInt32 maxSecurityTokenLifetime; // in ms

– ограничения для безопасных каналов;

UA_UInt16 maxSessions;

UA_Double maxSessionTimeout; /* in ms */

– ограничения для сессий.

Традиционно применяется следующий способ создания конфигурации сервера:

- сначала создать структуру конфигурации и инициализировать ее по умолчанию;
- модифицировать те поля, которые важны в конкретной задаче (например, добавить сертификат сервера);
- инициализировать сервер с использо-ванием отредактированной конфигурации.

В головной функции main() серверного приложения конфигурация сервера сформирована в результате следующих действий:

1) определены и инициализированы нулями структуры данных для сертификата, закрытого ключа и для конфигурации сервера: UA_ByteString certificate, UA_ByteString privateKey, UA_ServerConfig config_, *config=&config_;

2) объявлен и инициализирован статический массив реквизитов доступа static UA_UsernamePasswordLogin logins[3];

3) сертификат и закрытый ключ загружены из файловой системы tar;

4) структура конфигурации сервера заполнена значениями по умолчанию с добавлением сертификата, закрытого ключа и списка реквизитов доступа посредством API-функции UA_ServerConfig_setDefaultWithSecurityPolicies (). Списки доверенных и отзываемых сертификатов оставлены пустыми:

```
retval =
UA_ServerConfig_setDefaultWithSecurityPolicies(config, 4840,
&certificate, &privateKey,
trustList, trustListSize,
issuerList, issuerListSize,
revocationList, revocationListSize);
```

(4840 – порт, используемый по умолчанию для TCP-соединения сервера OPC UA);

5) в поле Description URI приложения приведено в соответствие с данными сертификата:

```
config->applicationDescription.applicationUri
=
UA_String_fromChars("urn:open62541.server.application");
```

6) установлен режим пользовательского доступа по логину и паролю (вызов API-функции UA_AccessControl_default). Список пользователей и паролей указан в массиве logins, передаваемом среди аргументов функции:

```
config->accessControl.clear(&config->
```

accessControl);
retval = UA_AccessControl_default(config, false, NULL,&config->securityPolicies[config->
securityPoliciesSize-1].policyUri, 3, logins);
7) создан экземпляр сервера с отредактированной конфигурацией:
UA_Server *server =
UA_Server_newWithConfig(config);

5.3 Создание информационной модели в адресном пространстве сервера

Функция static void createCustomInheritance (UA_Server *server) создает в адресном пространстве следующие узлы:

- объекты типа «папка» с именами DI, AI как компоненты системной папки Objectcs;
- по четыре компонента каждого объекта: переменные с именами DI0, DI1,DI2, DI3 с типом данных UA_BOOLEAN (AI0, AI1,AI2, AI3 с типом данных UA_INT32) с доступом по чтению и записи;
- ссылки типа «hasComponent» между объектами и их компонентами.

Для создания объектов и переменных вызываются функции:

UA_Server_addObjectNode();
UA_Server_addVariableNode().

При выполнении этих функции создаются соответствующие ссылки.

Вызову функций, создающих узел, предшествует создание и инициализация структур, определяющих атрибуты узла:

UA_ObjectTypeAttributes otAttr;
UA_ObjectAttributes oAttr;
UA_VariableAttributes vAttr;

Создается структура для идентификатора будущего узла:

UA_NodeId df_id;

Структуры атрибутов будущего узла инициализируются значениями по умолчанию, затем редактируются поля «description» и «displayName», например:

oAttr = UA_ObjectAttributes_default;
oAttr.description =
UA_LOCALIZEDTEXT("en-US", "DI");
oAttr.displayName =
UA_LOCALIZEDTEXT("en-US", "DI");

Для переменных также редактируются атрибуты доступа и типа данных, например:

vAttr.accessLevel =
UA_ACCESSLEVELMASK_READ |
UA_ACCESSLEVELMASK_WRITE;
vAttr.dataType
=UA_TYPES[UA_TYPES_UINT32].typeId;

Рассмотрим параметры функции UA_Server_addObjectNode() на примере ее вызова для создания объекта DI.

Интерфейс функции:

UA_StatusCode UA_Server_addObjectNode
(
UA_Server *server,
constUA_NodeIdrequestedNewNodeId,
constUA_NodeIdparentNodeId,
constUA_NodeIdreferenceTypeId,
constUA_QualifiedNamebrowseName,
constUA_NodeIdtypeDefinition,
constUA_ObjectAttributesattr,
void *nodeContext,
UA_NodeId *outNewNodeId
)

Параметры:

- UA_Server *server – сервер,
- constUA_NodeIdrequestedNewNodeId – идентификатор создаваемого узла. Идентификатор можно указать явно или запросить новое значение у сервера. Если в качестве параметра указать числовой NodeId с числовым идентификатором 0, то будет выбран случайный свободный числовой NodeId в соответствующем пространстве имен. В примере: UA_NODEID_NUMERIC(1,0) – запрос на выбор свободного числового значения в пространстве имен сервера;
- constUA_NodeIdparentNodeId –идентификатор родительского узла. В примере: UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER) – числовой идентификатор папки Objects в системном пространстве имен;
- constUA_NodeIdreferenceTypeId – идентификатор типа ссылки родительского узла на создаваемый узел. В примере: UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT) – числовой идентификатор типа ссылки « hasComponent» в системном пространстве имен;
- constUA_QualifiedNamebrowseName – атрибут « browseName» создаваемого узла. В примере: UA_QUALIFIEDNAME(1, "DI") – имя "DI" в пространстве имен сервера;

constUA_NodeIdtypeDefinition – идентификатор типа создаваемого объекта. В примере:

UA_NODEID_NUMERIC(0,UA_NS0ID_FOLDERTYPE) – идентификатор стандартного типа объектов «папка» в системном пространстве имен;

- constUA_ObjectAttributes attr – атрибуты создаваемого узла. В примере передается ранее созданная, инициализированная и отредактированная структура oAttr;
- void *nodeContext – контекст узла (пользовательские данные, предназначенные для экспонирования в адресном пространстве). В примере

передается значение NULL;

- UA_NodeId *outNewNodeId - указатель на структуру идентификатора созданного узла. Идентификатор будет записан по этому указателю. В примере передается указатель на ранее созданную структуру (&df_id).

Параметры функции UA_Server_addVariableNode(), использованной в примере для создания узлов переменных, не отличаются по составу и смыслу от параметров функции UA_Server_addObjectNode(). Отметим только отличия в способе их задания при вызове функции в рассматриваемом примере:

- идентификатор создаваемого узла указывался явно как числовой идентификатор в пространстве имен сервера, например: UA_NODEID_NUMERIC(1, 30300);

- в качестве идентификатора родительского узла указывалась структура df_id, в которую был записан идентификатор родительского объекта при его создании;

- в качестве идентификатора типа создаваемого объекта был указан идентификатор базового типа переменной из системного адресного пространства: UA_NODEID_NUMERIC(0, UA_NS0ID_BASEDATAVARIABLETYPE);

- в качестве указателя на структуру идентификатора созданного узла указан NULL (возвращать идентификатор не нужно, поскольку он задан явно).

5.4 Связь переменных с источниками внешних данных

Сервер OPC UA предоставляет возможность организовать выполнение с заданной периодичностью пользовательских функций с прототипом:

typedef void(*UA_ServerCallback)
(UA_Server*server,void*data);

Это свойство использовано в серверном приложении для организации асинхронного обновления значений переменных адресного пространства сервера дискретными и аналоговыми сигналами, считываемыми с модулей ввода-вывода по шине Modbus.

Разработана пользовательская функция
static void
testCallback(UA_Server *server, void *data),
которая:

- считывает значения сигналов по шине Modbus средствами ОСРВ Багет 2.7;

- преобразует их тип и копирует значения в переменные адресного пространства с использованием функций БПП OPC UA UA_Variant_setScalar(), UA_Server_writeValue().

Головная функция серверного приложения main() содержит вызов функции БПП OPC UA UA_Server_addRepeatedCallback:

UA_Server_addRepeatedCallback(server, testCallback, NULL, 100, NULL);

Этот вызов обеспечивает периодическое выполнение функции testCallback с интервалом 100 мсек.

6 Заключение

Модельная задача продемонстрировала работоспособность библиотеки поддержки протокола OPC UA на отечественном оборудовании под управлением отечественной операционной системы реального времени. Представляется перспективным развивать эту библиотеку. В частности, желательно расширить состав поддерживаемых сервисов, дополнив его сервисами доступа к историческим данным и поддержкой событий.

Публикация выполнена в рамках государственного задания по проведению фундаментальных исследований по теме «Создание и реализация доверенных систем искусственного интеллекта, основанных на новых математических и алгоритмических методах, моделях быстрых вычислений, реализуемых на отечественных вычислительных системах» (FNEF-2024-0001)».

OPC UA Protocol Support Library for Programmable Logic Controllers of the Baguette Family

T. K. Gringauz, D. V. Yarikov

Abstract. The open62541 freely distributed library implementing the OPC UA protocol has been ported to the environment of the domestic real-time operating system of the Baget family. The article provides an overview of the basic concepts of OPC UA, lists the functional capabilities of the ported version of the library, describes a model task, and provides an analysis of key fragments of its code.

Keywords: programmable logic controller, OPC UA protocol, RTOS Baget

Литература

1. А.Н. Годунов, В.А. Солдатов. Операционные системы семейства Багет (сходства, отличия и перспективы). «Программирование», т.40 (2014), № 5, 68 – 76.

2. OPC Unified Architecture Specification. Release 1.04. November 22, 2017. URL: https://reference.opcfoundation.org/ (дата обращения – 31.10.2024).

3. А.А. Титаев. Промышленные сети: учебное пособие, Екатеринбург, издательство Уральского университета, 2020.

4. open62541 Documentation. Release 1.0.5-1-g982f0796. January 05, 2021. URL: https://www.open62541.org/doc/open62541-1.0.pdf (дата обращения – 31.10.2024).

5. open62541 Documentation. Release 1.4. URL: https://www.open62541.org/doc/master/index.html (дата обращения – 31.10.2024).

6. Unified Automation. C++ UA Server SDK Documentation. Release 1.5.2.336. URL: https://documentation.unified-automation.com/uasdkcpp/1.5.2/html/index.html (дата обращения – 31.10.2024).

Разработка приложений для ПЛК на основе принципов стандарта ARINC 653

А.Н. Онин[1]

[1]НИЦ «Курчатовский институт» – НИИСИ, Москва, Россия, alexii@niisi.ras.ru.

Аннотация. В статье рассматривается использование принципов стандарта ARINC 653 как основы для разработки приложений на языках стандарта МЭК 61131-3 для исполнения на программируемом логическом контроллере (ПЛК), функционирующем под управлением ОСРВ Багет 3.6. Рассматривается схема использования многопоточного подхода реализации прикладных задач, когда каждая задача функционирует со своим периодом и приоритетом, независимо от других прикладных задач и программ-драйверов обслуживающих устройства ввода-вывода. В статье также рассматриваются принципы формирования диагностических сообщений.

Ключевые слова: ПЛК, ОС РВ Багет 3.6, МЭК 61131-3, ARINC 653, система диагностики

1. Введение

В статье рассматривается методы разработки приложений для программируемых логических контролеров (ПЛК) [1], написанных на языках стандарта МЭК 61131-3 [2, 6] и предназначенных для исполнения в среде, поддерживающей стандарт ARINC 653 [3]. В статье приведено описание принципов функционирования задач на ПЛК, поддерживающих стандарт ARINC 653, приведены основные характеристики и отличия от аналогов.

Рассматриваемые в статье принципы функционирования задач на ПЛК (далее будем называть это схемой функционирования) отличается от типовой схемы, в которой ПЛК имеет один процессор и выполняет несколько задач псевдопараллельно, последовательными порциями [4].

Типовая схема функционирования задач на ПЛК основана на следующих принципах:

- организуется один поток управления (диспетчер);
- задаётся длительность основного цикла (такт). Такт представляет собой целое количество миллисекунд (мс);
- каждая задача имеет период. Период задачи должен быть кратен такту;
- длительность такта зависит от периодов функционирования прикладных задач, и равен наибольшему кратному делителю периодов всех задач (ниже будет приведён пример);
- диспетчер в начале цикла получает входные данные от устройств ввода и записывает их во входные переменные задач;
- диспетчер определяет, какие задачи должны выполняться на текущем такте, и запускает их;
- диспетчер в конце цикла извлекает данные из выходных переменных задач и отправляет их в устройства вывода.

К особенностям типовой схемы можно отнести следующие:

- длительность такта зависит от периодов функционирования прикладных задач, и может оказаться намного меньше минимального периода задач. Приведём пример: на ПЛК должны функционировать три задачи с периодами 10, 15 и 32 мс. При этом наибольший кратный делитель периодов всех задач будет равен 1 мс, таким образом длительность такта будет равна 1 мс, что в 10 раз меньше периода задачи с минимальным периодом. В результате диспетчер будет в 10 раз чаще обращаться к устройствам ввода-вывода, чем это требуется для решения задач. Кроме того, задача может обрабатывать данные более 1 мс, а это больше периода такта, что приведёт к общему замедлению обработки данных на ПЛК;
- поскольку все задачи, а также программы обращения к устройствам ввода-вывода, выполняются в одном потоке, то сбой в одной из задач, или зависание на приёме данных от входного устройства, может привести к неработоспособности всех задач на ПЛК.

Предлагаемая схема функционирования задач на ПЛК с использованием стандарта ARINC 653 имеет преимущество по сравнению с типовой схемой.

2. Описание схемы функционирования

ПЛК функционирует циклически, выполняя в основном цикле предписанные для цикла действия. Цикл начинается со сбора данных с модулей ввода, затем исполняется прикладная программа ПЛК и заканчивается цикл выводом данных в устройства вывода. Во время сбора данных с модулей ввода, и посылки данных в устройства вывода, прикладная программа находится в состоянии ожидания завершения операций ввода-вывода. Прикладная программа обращается только к своим и глобальным переменным, и не взаимодействует с внешним оборудованием, устройствами ввода и вывода. В соответствии с перечисленными выше положениями прикладная программа может функционировать в операционной системе, построенной на принципах стандарта ARINC 653.

2.1. Использование принципов стандарта ARINC 653

В качестве операционной системы, функционирующей на ПЛК, рассмотрим ОСРВ Багет 3.6 [5], которая поддерживает выполнение прикладных программ в изолированных ARINC-partition (ARINC-процессах) [3, 5]. Связь прикладных программам (задач), функционирующих в ARINC-процессах, с внешним оборудованием происходит через системный POSIX-partition (POSIX-процесс, ядро операционной системы) [5]. Для передачи значений каждой переменной задачи, связанной с входными или выходными данными устройства, между ARINC-процессом и POSIX-процессом организуется Sampling-канал [3, 5]. Если переменная задачи связана с типом данных MEMORY [6], то есть является как входной, так и выходной, тогда для такой переменной организуются два разнонаправленных Sampling-канала (входной и выходной). Далее по тексту множество Sampling-каналов будем назвать буферной зоной.

Каждая задача, выполняющая прикладные программы, написанные на языках стандарта МЭК 61131-3 [6], представляет собой отдельный ARINC-process (поток управления) [3, 5], под управлением которого задача выполнятся циклически, с заданным периодом и приоритетом. При этом периоды выполнения задач не обязаны быть кратны друг другу.

В соответствии со стандартом ARINC 653 в функционирующей системе должен быть установлен основной цикл выполнения всех потоков управления. Этот цикл называется «Major Frame» [3, 5]. Длительность периода Major Frame может быть любой, при этом необходимо учитывать следующие обстоятельства:

- длительность Major Frame должна быть больше времени решения самой длительной задачи;

- длительность Major Frame не должна быть слишком большой, поскольку необходимо обеспечить в ПЛК поддержку возможности безударного переключения:

а) - на резервный модуль, в случае обнаружения сбоя программы или оборудования на основном модуле;

б) - на обновлённую программу, при «горячем» обновлении прикладного программного обеспечения на ПЛК.

Периоды циклических задач не обязаны быть кратными длительности периода Major Frame. При этом, по окончании периода Major Frame все задачи должны завершить свои циклы обработки данных. В случае, если задача не успеет завершить обработку данных по окончании периода Major Frame, то произойдёт исключительная ситуация, обработчик которой отправит соответствующее сообщение в систему диагностики, и задача продолжит своё выполнение в следующий период Major Frame, в соответствии со своим приоритетом [5].

2.2. Синхронизация управления

Функционирование ПЛК состоит в выполнении циклических и разовых задач. Циклические задачи выполняются периодически. Разовые задачи выполняются по событию. Событие – изменение по переднему фронту заданной пользователем переменной типа BOOL. Переменная должна устанавливаться в значение TRUE в контексте другой прикладной задачи.

Каждая задача обладает приоритетом. Приоритеты задач устанавливаются в соответствии со стандартом МЭК 61131-3, в диапазоне от 0 до 31. Задачи, у которых приоритет меньше, выполняются раньше.

Циклическая задача обладает периодом. Период — это промежуток времени, через который задача должна быть запущена снова. Операционная система планирует выполнение периодических задач в соответствии с заданной конфигурацией периодов потоков управления [5].

Прикладная программа, согласно МЭК 61131-3, не взаимодействует с внешними устройствами, все входные и выходные данные сохраняются в переменных программы. В начале цикла входные данные, полученные от входных устройств, загружаются из POSIX-процесса во входные переменные программы. В конце цикла значения выходных переменных программы отправляются в POSIX-процесс для формирования выходных данных, которые отправляются в выходные устройства. Загрузка и выгрузка данных между ARINC-процессом и

POSIX-процессом осуществляется через буферную зону.

В POSIX-процессе для каждого обслуживаемого устройства создаётся свой циклический поток управления с заданными периодом и приоритетом. Поток управления на каждом цикле выполняет следующие действия:

- выполняет инициализацию устройства в случае, если устройство не было инициализировано;
- получает от устройства входные данные;
- отправляет полученные входные данные через буферную зону задачам в раздел ARINC;
- читает выходные данные задач, сформированные в буферной зоне;
- отправляет выходные данные в устройство.

Потоки управления обслуживающие устройства работают независимо и не синхронизированы между собой.

В случае обнаружения ошибки при обращении к устройству ввода-вывода будет установлен признак, что устройство неработоспособно, и на следующем цикле потока управления обслуживания устройства будет выполнена попытка инициализировать устройство заново. Таким образом, осуществляется программная поддержка резервирования модулей ввода-вывода.

2.3 Глобальные переменные МЭК-программ

Задачи вызывают программы, которые могут обращаться к глобальным переменным. Каждая задача выполняется в своём потоке управления, таким образом может образоваться конкурентный доступ к глобальным переменным. Чтобы избежать одновременного обращения к глобальной переменной из двух или более задач, в задачах используется преимущественная блокировка [3]. Это означает, что задача выполняет свой цикл непрерывно, за счёт того, что в начале цикла устанавливается блокировка, и в конце цикла блокировка снимается. Для разовых задач блокировка устанавливается на всё время выполнения обработки данных задачи.

2.4 Принципы формирования диагностических сообщений

Система диагностики может быть реализована в виде библиотеки функций, которые будут вызывать прикладные программы, драйверы обслуживающие устройства ввода-вывода, программы синхронизации и системные компоненты.

Система диагностики поддерживает несколько типов диагностических сообщений:

- неисправимый сбой. При этом система диагностики автоматически запускает процедуру перехода функционирования задач на резервный модуль;
- критическая ошибка. При этом требуется немедленное исправление ошибки или необходимо выполнить переход на резервный модуль;
- ошибка. При этом считается, что программа самостоятельно исправила ошибку и возможно продолжение функционирования задачи;
- предупреждение. Программа сообщает о нештатной ситуации, которая не приводит к сбою функционирования;
- уведомление. Программа сообщает о важном событии;
- информирование. Такие сообщения формируются программой при штатном функционировании задач;
- статистика. Программа сообщает о статистических показателях, накопленных во время функционирования задач;
- отладка. Этот тип предназначен для вывода отладочных сообщений.

В результате вызова функций системы диагностики функции формируют текстовые сообщения, которые сохраняются в буфере в оперативной памяти. Кроме сохранения сообщений в буфере система диагностики отправляет сообщения в консоль операционной системы [5].

Система диагностики обеспечивает передачу сообщений из ARINC-процесса в POSIX-процесс по каналам с очередью сообщений [3, 5]. В POSIX-процессе сообщения, полученные из ARINC-процессов и POSIX-процесса, накапливаются в кольцевом буфере.

С целью долговременного хранения сообщений в системе диагностики функционирует поток управления, который копирует сообщения из кольцевого буфера в файловую систему. Файлы сообщений организованы следующим образом:

- название файла содержит дату и время первого сообщения, которое сохраняется в файле;
- название файла содержит тип диагностического сообщения;
- в файле сохраняются множество сообщений одного типа. Таким образом, происходит естественная фильтрация сообщений по типам, что упрощает обнаружение критически важных сообщений;
- сообщения накапливаются в файле пока размер файла не превышает 1 Мбайт. Небольшой размер файла позволяет быстро загружать его из ПЛК на инженерную станцию;

Новый файл с диагностическими сообщениями создаётся в следующих случаях:

- произошло превышение размера файла в 1 Мбайт;
- наступили новые сутки. Это способствует быстрому поиску сообщений по дате;
- произошёл перезапуск задач на ПЛК. Это

способствует быстрому поиску сообщений, которые возникали в определённый запуск ПЛК.

3. Заключение

Рассмотренный подход к разработке приложений для ПЛК на основе принципов стандарта ARINC 653 соответствует принципам стандарта МЭК 61131-3, и обладает следующими положительными свойствами:

- многопоточная реализация прикладных задач и программ обслуживания устройств ввода-вывода повышает надёжность функционирования ПЛК за счёт независимого выполнения прикладных задач от программ обслуживающих устройства. В случае сбоя одного или нескольких компонент (задачи, устройства), другие компоненты продолжат своё функционирование, а система диагностики сообщит о произошедших сбоях;

- использование системы диагностики позволяет автоматизировать сбор диагностических и статистических данных с сохранением их в файлах файловой системы, а также автоматизировать реакцию на критически важные события.

- в прикладных задачах используется преимущественная блокировка, которая обладает положительными и отрицательными свойствами. Отрицательным свойством преимущественной блокировки является следующее: в то время, пока одна задача не завершит свой цикл обработки данных, другие задачи, в том числе и более приоритетные, будут находиться в состоянии ожидания. Положительным свойством преимущественной блокировки является то, что не будет создаваться конкурентный доступ к глобальным переменным, а также будет исключена возможность прерывания процедуры обработки данных выполняемой задачи. На практике преимущественная блокировка реализована как опциональная возможность.

Рассмотренный подход к разработке приложений для ПЛК на основе принципов стандарта ARINC 653 был успешно реализован в решении задач на одном из образцов ПЛК-2.

Публикация выполнена в рамках государственного задания по проведению фундаментальных исследований по теме «Создание и реализация доверенных систем искусственного интеллекта, основанных на новых математических и алгоритмических методах, моделях быстрых вычислений, реализуемых на отечественных вычислительных системах» (FNEF-2024-0001)».

Programming IEC 61131-3 Tasks for PLC Running under RTOS Baget 3.6 Based on ARINC 653 Standard

A. Onin

Abstract. The article discusses the use of the ARINC 653 standard principles as a basis for developing applications in the languages of the IEC 61131-3 standard for execution on a programmable logic controller (PLC) running under the RTOS Baget 3.6. The article discusses the scheme for using a multi-threaded approach to implementing application tasks, when each task operates with its own period and priority, independently of other application tasks and driver programs servicing input-output devices. The article also discusses the principles of generating diagnostic messages.

Keywords: PLC, OS RV Baget 3.6, IEC 61131-3, ARINC 653, diagnostic system

Литература

1. Шишов О.В. Программируемые логические контроллеры: учебник / О.В. Шишов. — Москва : ИНФРА-М, 2024. — 461 с. — (Высшее образование). — DOI 10.12737/2030899. - ISBN 978-5-16-018581-1.

2. ГОСТ Р МЭК 61131-3-2016.

3. ARINC 653 API and its application – An insight into Avionics System Case Study. Ananda C.M., Sabitha Nair, and Mainak G.H. CSIR-National Aerospace Laboratories, Bangalore–560 017, India. Defence Science Journal, Vol. 63, No. 2, March 2013, pp. 223-229.

4. Петров И.В. Программируемые контроллеры. Стандартные языки и приемы прикладного проектирования / – М.: СОЛОН-Пресс, 2004.

5. Годунов А.Н., Солдатов В.А. Спецификация ARINC 653 и её реализация в операционной системе реального времени Багет 3, НИИСИ РАН г. Москва, 2015 № 5, с. 3-17.

6. Романов С. Изучаем Structured Text МЭК 61131-3. ISBN: 978-1-64199-106-3 2020.

Сокращение временных затрат на проектирование СФ-блоков при использовании автоматизации подбора входных параметров

Е.С. Кочева[1], Н.В. Желудков[2], Е.В. Ткаченко[3], Б.Е. Евлампиев[4], К.А. Петров[5]

[1]НИЦ "Курчатовский институт" - НИИСИ, Москва, Россия,
kocheva@cs.niisi.ras.ru;

[2]НИЦ "Курчатовский институт" - НИИСИ, Москва, Россия,
nvgel@cs.niisi.ras.ru;

[3]НИЦ "Курчатовский институт" - НИИСИ, Москва, Россия,
etkachenko@cs.niisi.ras.ru;

[4]НИЦ "Курчатовский институт" - НИИСИ, Москва, Россия,
eboris@cs.niisi.ras.ru;

[5]НИЦ "Курчатовский институт" - НИИСИ, Москва, Россия,
petrovk@cs.niisi.ras.ru.

Аннотация. Проведен сравнительный анализ времени, затрачиваемого на проектирование цифровых сложно-функциональных блоков с применением автоматизированного подбора входных параметров в САПР и без автоматизированного подбора. Для проведения анализа использовались два сложно-функциональных блока: целочисленного умножения-деления и интерфейс ввода/вывода. Анализ показал преимущество автоматизированной системы подбора входных параметров САПР с точки зрения сокращения времени, затрачиваемого на поиск их оптимальных значений, на 46% и более.

Ключевые слова: топологическое проектирование, СБИС, СФ-блок, Optuna, оптимизация.

1. Введение

В современном мире скорость проектирования сложно-функциональных блоков (СФ-блоков) является критически важным параметром разработки, ее повышение означает повышение эффективности разработки, снижение затрат предприятия и дает таким образом конкурентное преимущество организации – дизайн-центру.

Процесс проектирования цифровых СФ-блоков в системах автоматического проектирования (САПР) можно разделить на 2 глобальных этапа: настройка и отладка маршрута проектирования, непосредственный запуск данного маршрута в САПР. В процессе настройки и отладки маршрута проектирования производится настройка целого ряда входных параметров, оказывающих непосредственное влияние на итоговое качество полученной топологии будущей микросхемы либо СФ-блока, а именно на такие ключевые выходные характеристики как площадь, потребляемая мощность, быстродействие и т.д.

Для того, чтобы в результате разработки было получено устройство, отвечающее всем заданным требованиям, одной из ключевых задач является непосредственно подбор оптимального набора значений входных параметров (НЗВП) САПР, среди которых определяется начальная плотность ячеек в СФ-блоке, период тактового сигнала, доступное количество слоёв металлизации для осуществления маршрутизации в СФ-блоке и другие.

Процесс подбора НЗВП довольно ресурсоемкий как с точки зрения трудозатрат, так и затрат по времени. Это обусловлено тем, что для определения наилучшего НЗВП для данного СФ-блока разработчику требуется вручную перебрать достаточно большое количество вариантов, провести анализ выходных характеристик, соответствующих каждому НЗВП, и только после возможно сделать вывод о качестве полученной топологии.

Решением данной проблемы является использование механизма автоматизированного подбора оптимального НЗВП СФ-блока для САПР. Данный подход призван существенно сократить необходимое количество итераций запуска маршрута проектирования, а также время, затрачиваемое на осуществление запусков, за счет их распараллеливания.

Для реализации данного подхода был выбран фреймворк Optuna [1], применяемый также в

рамках моделей машинного обучения для автоматизированного поиска оптимального НЗВП в кратчайшие сроки.

В рамках данной работы проведена сравнительная оценка сокращения времени проектирования СФ-блоков, затрачиваемого на подбор оптимального НЗВП за счет применения алгоритма автоматизированного подбора НЗВП с использованием фреймворка Optuna.

2. Автоматизированный подбор оптимальных значений входных параметров

В работе [2] был представлен и подробно описан принцип используемого в данной работе алгоритма, на базе фреймворка Optuna (рис. 1).

С помощью управляющего скрипта осуществляется формирование уникального НЗВП, с последующей записью в файл конфигураций, передаваемый в САПР при запуске унифицированного маршрута проектирования.

В рамках каждого запуска происходит формирование набора выходных метрик, содержащих информацию о выходных характеристиках полученной топологии цифрового СФ-блока. Для оценки качества топологии анализируются следующие выходные параметры: быстродействие, площадь, потребляемая мощность и количество DRC-нарушений.

В рамках алгоритма по завершению итерации формируется числовая оценка выбранных ключевых выходных характеристик спроектированного СФ-блока с помощью Score-функции [2]:

$$S = \alpha\, sigm(clk_n) + \beta\, sigm(pwr_n) + \gamma\, sigm(area_n) + \eta\, sigm(DRC_n),$$

где α, β, γ и η – коэффициенты влияния периода, потребляемой мощности, площади и числа DRC-нарушений соответственно; $sigm$ – сигмоидальная функция; clk_n, pwr_n, $area_n$ и DRC_n – нормированные значения соответствующих выходных параметров.

Рис. 1. Алгоритм автоматизированного подбора НЗВП

Алгоритм формирует уникальный НЗВП, на его основе осуществляет запуск САПР, анализирует результат в виде значения Score-функции, и осуществляет следующий запуск, основываясь на ранее полученных данных, тем самым довольно оперативно стремится к подбору оптимального НЗВП. В работе [3] было продемонстрировано, что данный метод также способствует достижению лучших характеристик для заданного СФ-блока, чем при проектировании без его использования.

3. СФ-блоки для проведения сравнительного анализа

С целью проведения сравнительного анализа времени, затрачиваемого на поиск оптимальных значений НЗВП разработчиком и времени, затрачиваемым при использовании описанного выше алгоритма автоматизированного подбора НЗВП для эксперимента были выбраны два цифровых СФ-блока, разработанные в НИЦ «Курчатовский институт» - НИИСИ, в соответствии с техническими требованиями для КМОП 65 нм:

- cp2k_pio – модуль, обеспечивающий взаимодействие между центральным процессором и сопроцессором. В его состав входит порядка 1000 элементов;
- int_mult_div – модуль, предназначенный для операций целочисленного умножения и деления

64-разрядных чисел. В его состав входит порядка 30 000 элементов.

За счет того, что они обладают небольшими размерами и содержат только элементы стандартных (библиотечных) ячеек, обеспечивается небольшое время для полного прохождения маршрута проектирования топологии способствуя более быстрому поиску оптимального НЗВП.

Для проведения экспериментального проектирования были определены десять входных параметров, оказывающих наибольшее влияние на ключевые выходные характеристики цифрового СФ-блока СБИС, совпадающие с представленными в работе [3].

Ввиду того, что при определении функции количественной оценки качества топологии все характеристики считаются равнозначно важными, коэффициенты влияния α, β, γ и η, при расчете Score-функции равны единице.

4. Результаты экспериментов

Посредством алгоритма для каждого из исследуемых СФ-блоков было пройдено по 200 итераций, каждая с уникальным НЗВП. Результаты для блоков cp2k_pio и int_mult_div представлены на рисунках 2 и 3 соответственно.

Рис. 2 – Результаты запуска фреймворка Optuna для СФ-блока cp2k_pio

Рис. 3 – Результаты запуска фреймворка Optuna для СФ-блока int_mult_div

Красная линия на графиках обозначает и отслеживает последнее достигнутое минимальное значение Score-функции.

Подробные результаты эксперимента приведены в таблице 1.

Таблица 1. НЗВП и выходные характеристики при минимальном значении Score-функции для СФ-блоков cp2k_pio и int_mult_div

Имя СФ-блока	**int_mult_div**	**cp2k_pio**
Номер итерации	189	118
НЗВП		
Период тактового сигнала, нс	1,10	1,45
Максимальный фронт сигналов, нс	0,25	0,15
Плотность заполнения ячейками, %	0,45	0,65
Верхний металл для разводки	7	7
Ограничение на максимальную длину тактового сигнала	Да	Да
Максимальная длина дерева тактового сигнала, нс	1,5	0,5
Особые правила для цепей тактового сигнала	Нет	Нет
Экранирование цепей тактового сигнала	Нет	Да
Типы ячеек с разным пороговым напряжением	lvt, hvt	svt, hvt
Типы ячеек дерева тактового сигнала	lvt	lvt
Выходные характеристики		
Значение Score-функции	6,27	5,73
Частота, МГц	886,5	995
Площадь, мкм^2	315 302	6 809
Потребляемая мощность, мВт	304	4,8
Количество DRC-нарушений	0	0

В рамках 200 запусков минимальное значение Score-функции для СФ-блока int_mult_div было достигнуто на 189 итерации. При этом

были достигнуты следующие значения ключевых выходных характеристик:

- частота составила ~887 МГц;
- потребляемая мощность ~806 мВт;
- площадь дизайна ~315302 мкм^2;
- DRC-нарушения отсутствуют.

Для СФ-блока cp2k_pio минимальное значение Score-функции не изменялось с итерации 118. При этом значения ключевых выходных характеристик:

- частота составила ~995 МГц;
- потребляемая мощность ~5 мВт;
- площадь дизайна ~6809 мкм^2;
- DRC-нарушения отсутствуют.

В таблице 2 приведена сравнительная оценка затрат времени при проектировании СФ-блоков cp2k_pio и int_mult_div.

Таблица 2. Сравнение затрат времени на подбор оптимального НЗВП для СФ-блоков cp2k_pio и int_mult_div

Имя СФ-блока	**int_mult_div**	**cp2k_pio**
Количество возможных комбинации НЗВП	698 544	
Средняя продолжительность одной итерации	~2 ч. 20 мин.	~12 мин.
Время для перебора всех вариантов НЗВП	~1 606 651 ч.	~138 661 ч.
Количество запущенных итераций	200	
Суммарное время всех итераций	~554 ч.	~50 ч.
Реальное время (с учетом распараллеливания запусков)	86 ч. 36 мин.	27 ч. 15 мин.

Согласно полученным данным, СФ-блок int_mult_div показывает более явную сходимость к наименьшему значению Score-функции (рис. 3). Предположительно это может быть связано с тем, что данный СФ-блок обладает большими размерами в сравнении с СФ-блоком cp2k_pio, а значит и изменения в НЗВП на него сказываются более значительно.

Обращаясь к таблице 1, можно увидеть, что в рамках 200 запусков с помощью автоматизированной системы подбора оптимального НЗВП удалось добиться высоких показателей выходных характеристик для каждого из СФ-блоков.

Согласно таблице 2, автоматизированный поиск оптимального набора значений входных параметров САПР с учетом параллельных запусков, позволяет сократить время на 46% для блока cp2k_pio и на 84% для блока int_mult_div по сравнению с последовательными запусками. В дополнение к этому необходимо отметить, что данный метод производит поиск нужных параметров без участия разработчика, а следовательно, без привязки к его рабочему времени и количеству потенциально возможных запусков маршрута в день, что также существенно повышает эффективность разработки.

4. Заключение

В работе исследованы затраты времени, затрачиваемого на подбор оптимальных значений входных параметров САПР при проектировании топологии цифровых СФ-блоков как разработчиком, так и с применением автоматизированной системы подбора НЗВП на базе фреймворка Optuna, и проведен их сравнительный анализ.

Сравнительный анализ показал, что при использовании автоматизации подбора оптимальных входных параметров значительно (на 46% и более) сокращается время, затрачиваемое разработчиком на запуски маршрута проектирования СБИС в рамках подбора оптимального НЗВП.

Дальнейшие исследования требуют вопросы увеличения экономии времени разработчика, при условии роста размеров разрабатываемого СФ-блока, а также его архитектурной сложности. Отдельного рассмотрения требует автоматизация разработки СФ-блоков, в состав которых входят макроблоки (памяти, заказные СФ-блоки и пр.).

Публикация выполнена в рамках государственного задания НИЦ "Курчатовский институт" - НИИСИ по теме FNEF-2024-0003.

Reduction of Time Consumption on VLSI IP-blocks Designing Using Input Parameter Selection Automation

E.S. Kocheva, N.V. Zheludkov, E.V. Tkachenko, B.E. Evlampiev, K.A. Petrov

Abstract. A comparative analysis of the time spent on designing digital IP cores using automated selection of input parameters in CAD and without automated selection is carried out. Two IP cores were used for the analysis: Integer Multiplication/Division and the I/O interface. The result of the analysis showed the advantage of an automated system for selecting CAD input parameters in terms of reducing the time spent on searching for their optimal values.

Keywords:layout design, VLSI, IP-block, Optuna, optimization

Литература

1. Akiba T., Sano S., Yanase T., Ohta T., Koyama M. Optuna: A Next-generation Hyperparameter Optimization Framework // The 25th ACM SIGKDD International Conference, 2019. P. 2623-2631.
2. Желудков Н.В., Кочева Е.С., Евлампиев Б.Е., Ткаченко Е.В. Автоматизация поиска оптимальных входных параметров при проектировании СБИС // Наноиндустрия. – 2024. – Т. 17, № S10-1(128). – С. 246-250.
3. Кочева Е.С., Желудков Н.В., Ткаченко Е.В., Чумаков К.А., Петров К.А. Сокращение энергопотребления СФ-блоков посредством автоматизированного подбора оптимальных параметров проектирования // Труды научно-исследовательского института системных исследований Российской академии наук. – 2023. – Т. 13, № 4. – С. 80-84.

Поточечная запись информации в резисторную матрицу

В.Б. Котов[1], Г.А. Бесхлебнова[2]

[1] НИЦ «Курчатовский институт» – НИИСИ, Москва, Россия, kotov.vlb@yandex.ru;

[2] НИЦ «Курчатовский институт» – НИИСИ, Москва, Россия, gab19@list.ru

Аннотация. Для большой резисторной матрицы типа кроссбарр локальная запись информации в выбранный резистор (то есть изменение проводимости этого резистора) сталкивается с трудностями, связанными с ограниченным числом управляющих сигналов – напряжений на проводниках структуры. Поскольку число проводников значительно меньше числа резисторов, при подаче напряжения на целевой резистор, возникают напряжения на многих нецелевых резисторах. Соответствующие изменения проводимостей нецелевых резисторов необходимо компенсировать. В работе рассмотрена процедура записи с использованием высокочастотных гармонических сигналов и с управлением адресации с помощью варьирования сопротивлений резисторов подключения. На основе анализа с использованием модели простого резисторного элемента показана возможность точечной записи информации в резисторную матрицу, то есть локального изменения матрицы проводимостей. Обсуждаются условия, обеспечивающие выполнимость и удобство такой процедуры.

Ключевые слова: переменный резистор, высокочастотный сигнал, резисторная матрица, матрица проводимостей, поточечная запись

1. Введение

Хранение информации является наиболее очевидным применением массива переменных резисторов («мемристоров») [1-3]. Под переменным резистором подразумеваем резистор, сопротивление (проводимость) которого изменяется под действием протекающего тока. Резисторные структуры типа кроссбар легко изготовить. Считывание информации, то есть измерение проводимостей переменных резисторов, не вызывает затруднений [4]. Проблемы возникают при записи информации, то есть при изменении проводимостей резисторов. Желательно иметь возможность изменять произвольно (в определённых пределах) проводимость выбранного (целевого) резистора. Однако для больших массивов (матриц) переменных резисторов организовать индивидуальный доступ к каждому резистору нереально из-за огромного числа элементов. В регулярных структурах типа кроссбар каждый резистор соединяет два проводника. К каждому проводнику присоединено много резисторов (строка или столбец массива). Число управляющих сигналов – напряжений на проводниках – много меньше числа резисторов. Поэтому при подаче напряжения на целевой резистор возникают токи через многие нецелевые резисторы, что приводит к нежелательному изменению проводимостей этих резисторов. Необходимо как-то компенсировать это изменение.

Среди переменных резисторов наиболее популярны однонаправленные резисторы, для которых положительный ток (направление тока совпадает с направлением резистора) стремится увеличить проводимость резистора, а отрицательный ток – уменьшить. Для записи информации в таких резисторах обычно используются токи и напряжения постоянной полярности. При этом различные нецелевые резисторы испытывают разные по знаку изменения проводимости [4,5]. Компенсировать такие изменения затруднительно. Было бы удобнее работать с ненаправленными резисторами, изменение проводимости которых не зависит от направления тока. Однако подобные резисторы не пользуются популярностью. К счастью, аналогичного результата можно добиться при использовании однонаправленных резисторов, если для записи информации использовать переменный (высокочастотный) сигнал [6].

Возможны различные варианты процедуры записи с компенсацией нежелательных изменений матрицы проводимостей резисторов [7]. В данной работе рассмотрим запись информации в массив переменных резисторов типа кроссбар с помощью высокочастотных гармонических сигналов и с адресацией, осуществляемой с помощью выбора сопротивлений резисторов подключения, через которые на проводники кроссбара подаются напряжения. Предполагаем, что переменные резисторы однонаправленные и описываются моделью простого резисторного элемента [7].

На рисунке 1 изображена электрическая схема кроссбара с источниками напряжения V^i (i=1,…,m), V_j (j=1,…,n) и резисторами подключе-

ния r^i (i=1,…,m), r_j (j=1,…,n). $m\times n$ – размер резисторной матрицы. Эта матрица состоит из переменных резисторов R^i_j (i=1,…,m, j=1,…,n).

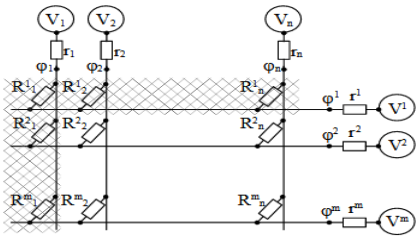

Рис. 1. Электрическая схема кроссбара с источниками напряжения и резисторами подключения

Резистор R^i_j находится на пересечении i-го горизонтального проводника и j-го вертикального проводника. Вообще верхний индекс («i») используем для нумерации горизонтальных проводников (или строк матрицы), а нижний индекс «j» – для нумерации вертикальных проводников (или столбцов матрицы). Сопротивления резисторов обозначаем так же, как и сами резисторы. Проводимость резистора, ток через него и напряжение на резисторе обозначаем буквами G, I, u соответственно.

2. Уравнения записи

Проводимость $G=1/R$ простого резисторного элемента выражается через единственную переменную состояния x: $G=G(x)$. Считаем, что переменная состояния меняется от 0 до 1. Состояние x=0 – это состояние с максимальным сопротивлением (основное состояние), а состояние x=1 соответствует минимальному сопротивлению резистора. Изменение переменной состояния описывается уравнением [7]

$$\frac{dx}{dt} = F(x,I), \quad (1)$$

где I – ток через резистор.

Используем следующее представление [7] для функции F:

$$F(x,I) = F_0(x) + F_x^+(x)F_I^+(I) + F_x^-(x)F_I^-(I) \quad (2)$$

Функции $F_0(x), F_x^{\pm}(x), F_I^{\pm}(I)$ наряду с функцией G(x) называем характеристическими функциями резистора. Первый член в правой части равенства (2) (отрицательный) отвечает за самопроизвольную релаксацию к основному состоянию x=0. Второй член (положительный при I>0 и равный нулю при I≤0) соответствует тенденции увеличения переменной x, когда направление тока совпадает с направлением резистора. Третий член (отрицательный при I<0 и равный нулю при I≥0) описывает ускоренную релаксацию к основному состоянию, когда направление тока противоположно направлению резистора.

Наиболее типичный вид характеристических функций $F_I^{\pm}(I)$ – степенной на соответствующих полуосях:

$$F_I^+(I) = A_+\theta(I)I^{\alpha_+}, F_I^-(I) = -A_-\theta(-I)(-I)^{\alpha_-} \quad (3)$$

с положительными коэффициентами $A_{\pm}$ и показателями $\alpha_{\pm}$.

Характеристические функции $F_0(x), F_x^{\pm}(x)$ определяют скорости записи/стирания в зависимости от текущего состояния резистора. Естественно считать, что они непрерывные на отрезке [0,1], положительные в интервале (0,1), и выполняются соотношения

$$F_0(0) = 0, F_x^+(0) > 0, F_x^+(1) = 0, \quad F_x^-(0) = 0, F_x^-(1) > 0. \quad (4)$$

Пусть на переменный резистор подаётся высокочастотный сигнал. Считаем, что период колебаний достаточно мал, так что за период состояние резистора меняется незначительно. С точки зрения электрических колебаний резисторы ведут себя как постоянные. Медленное изменение состояния каждого резистора описывается уравнением [6]

$$\frac{dx}{dt} = F_0(x) + M^+F_x^+(x) + M^-F_x^-(x) \quad (5)$$

где $M^+ = \langle F_I^+(I)\rangle, M^- = \langle F_I^-(I)\rangle$, угловые скобки обозначают усреднение по периоду колебаний. Для краткости опускаем индексы, нумерующие резисторы. Заметим, что величины M^+, M^- могут медленно (по сравнению с электрическими колебаниями) меняться со временем.

Для гармонического тока $I(t) = I_0\sin(\omega t + \varphi_0)$($I_0$ – амплитуда, ω – круговая частота, φ_0 – начальная фаза) при степенных характеристических функциях (3) получаем

$$M^+ = \frac{\kappa(\alpha_+)}{2\pi}A^+I_0^{\alpha_+}, M^- = -\frac{\kappa(\alpha_-)}{2\pi}A^-I_0^{\alpha_-} \quad (6)$$

где $\kappa(\alpha) = \int_0^{\pi}(\sin\varphi)^{\alpha}\,d\varphi$ – число порядка единицы при реальных значениях величины α. В частности, κ(1)=2, κ(2)=π/2, κ(3)=4/3. Величины M^+, M^- зависят от амплитуды тока, но не зависят от фазы и частоты сигнала.

Знак правой части уравнения (5), которую обозначим H(x), определяет направление изменения переменной состояния x: при H>0 величина x растёт со временем, при H<0 – уменьшается. Состояние x монотонно стремится к устойчивому стационарному состоянию (стационарной точке), которое получается при равенстве нулю правой части уравнения (6):

$$H(x) \equiv F_0(x) + F_x^+(x)M^+ + F_x^-(x)M^- = 0. \quad (7)$$

Заметим, что уравнение (6) пригодно для описания изменения состояния резистора под действием постоянного или медленно меняющегося тока. В этом случае, очевидно,

$$M^+ = F_I^+(I), M^- = F_I^-(I).$$

Пусть запись производится при подаче гармонических сигналов с частотой ω. В дальнейшем для простоты считаем, что запись производится в линейном режиме. Это означает, что для всех резисторов (одинаковых по своим характеристикам) выбрано базовое состояние x_b. Информация записывается в виде малых отклонений от базового состояния. Базовое состояние для всех резисторов массива легко сформировать, подав гармоническую разность потенциалов между всеми горизонтальными и всеми вертикальными проводниками. Амплитуду сигнала удобно выбрать так, чтобы соответствующее стационарное состояние совпадало с базовым состоянием x_b. При этом сопротивления резисторов подключения горизонтальных проводников должны быть одинаковыми, то же относится к резисторам подключения вертикальных проводников. Можно использовать и постоянную в течение некоторого времени разность потенциалов между горизонтальными и вертикальными проводниками.

Под действием гармонического напряжения

$$u(t) = \frac{1}{2}u_0 \exp(i\omega t) + \text{КС} \qquad (8)$$

(u_0 – комплексная амплитуда напряжения, КС – комплексно сопряжённое выражение) состояние резистора изменяется на величину

$$\Delta x = H(x_b)T, \quad (9)$$

где T – время записи, H определяется выражениями (6), (7), в которых $I_0=G(x_b)|u_0|$. При фиксированном времени записи T изменение состояния зависит только от вещественной амплитуды напряжения: $\Delta x = \Delta x(|u_0|)$. Эта зависимость универсальная, одинаковая для всех резисторов кроссбара. Осталось найти распределение напряжений на резисторах матрицы.

3. Распределение напряжений

Ток через резистор между i-ым горизонтальным проводником и j-ым вертикальным проводником I_j^i выражается через напряжение согласно закону Ома:

$$I_j^i = G_j^i u_j^i = G_j^i(\varphi^i - \varphi_j), \qquad (10)$$

где φ^i, φ_j – потенциалы i-го горизонтального и j-го вертикального проводников, которые выражаются через потенциалы источников V_i, V^j согласно уравнениям

$$\varphi^i = V^i - u^i, \varphi_i = u_i + \phi_i, \quad (11)$$

где u^i, u_j – напряжения на резисторах подключения r^i, r_j соответственно. Закон Ома для резисторов подключения даёт уравнения

$$I^i = g^i u^i, I_j = g_j u_j, \qquad (12)$$

причём I^i, I_j – токи, а $g^i=1/r^i$, $g_j=1/r_j$ – проводимости соответствующих резисторов.

Очевидно,

$$I^i = \sum_{j=1}^n I_j^i, I_j = \sum_{i=1}^m I_j^i. \qquad (13)$$

Из соотношений (10)-(13) получаем две системы уравнений для определения потенциалов проводников φ^i, $i = 1,\ldots,m$, и φ_j, $j = 1,\ldots,n$, при заданных потенциалах источников

$$\sum_{j=1}^m A^{ij}\varphi^j = b^i, i = 1,\ldots,m, \quad (14)$$

$$\sum_{j=1}^n A_{ij}\varphi_j = b_i, i = 1,\ldots,n, \quad (15)$$

где $A^{ij} = \delta^{ij} - \sum_{k=1}^n h_k^i v_k^i, A_{ij} = \delta_{ij} - \sum_{k=1}^m v_i^k h_j^k$,

$$b^i = \sum_{k=1}^n h_k^i \psi_k + \psi^i, b_i = \sum_{k=1}^m v_i^k \psi^k + \psi_i,$$

$$h_j^i = \frac{G_j^i}{g^i + G^i}, v_j^i = \frac{G_j^i}{g_j + G_j},$$

$$\psi^i = \frac{g^i}{g^i + G^i} V^i, \psi_i = \frac{g_i}{G_i + g_i} V_i,$$

$$G^i = \sum_{k=1}^n G_k^i, G_j = \sum_{k=1}^m G_j^k,$$

δ^{ij}, δ_{ij}– элементы единичных матриц размера $m\times m$ и $n\times n$ соответственно (символы Кронекера). Обе матрицы $[A^{ij}]$ и $[A_{ij}]$ невырожденные. Обозначая элементы соответствующих обратных матриц a^{ij} и a_{ij}, запишем решения систем (14) и (15) в виде

$$\varphi^i = \sum_{j=1}^m a^{ij} b^j, \varphi_j = \sum_{k=1}^n a_{jk} b_k. \quad (16)$$

Правые части равенств (16) являются линейными комбинациями потенциалов источников. Зная потенциалы проводников φ^i, φ_j, можно найти все токи и напряжения. В интересующем нас случае, когда

$$G_j^i = G, \quad (17)$$

решения (16) можно записать в явном виде:

$$\varphi^i = \frac{g^i V^i}{g^i + nG} + \frac{G}{g^i + nG}\frac{f_1 + f_0 f^1}{1 - f^0 f_0},$$
$$\varphi_j = \frac{g_j V_j}{g_j + mG} + \frac{G}{g_j + mG}\frac{f^1 + f^0 f_1}{1 - f^0 f_0}, \qquad (18)$$

причём

$$f^0 = \sum_{i=1}^m \frac{G}{g^i + nG}, f_0 = \sum_{j=1}^n \frac{G}{g_j + mG},$$

$$f^1 = \sum_{i=1}^m \frac{g^i V^i}{g^i + nG}, f_1 = \sum_{j=1}^n \frac{g_j V_j}{g_j + mG}.$$

Формулы (18) позволяют анализировать зависимость напряжений на переменных резисторах от сопротивлений резисторов подключения. Формулы (18) записаны для потенциалов и представляют линейную зависимость потенциалов проводников и напряжений на переменных резисторах от потенциалов (напряжений) источников. При использовании гармонических сигналов с заданной частотой такие же формулы справедливы для комплексных амплитуд соответствующих величин. В дальнейшем мы рассматриваем только такие сигналы (частный случай –

постоянные сигналы с нулевой частотой). Для краткости опускаем индекс «0» у комплексной амплитуды. Заметим, что в рамках принятого приближения комплексные амплитуды могут быть медленно меняющимися функциями времени.

Для случая прямого подключения проводников, когда $g^i \to \infty$, $g_j \to \infty$, получаем из (18)

$$\varphi^i = V^i, \varphi_j = V_j, u_j^i = V^i - V_j. \quad (19)$$

Резисторы подключения не оказывают влияние на распределение напряжений в силу своего отсутствия. В случае конечных, но больших значений проводимостей резисторов подключения, то есть при выполнении условий

$$g^i \gg nG, g_j \gg mG, \quad (20)$$

получаем простые выражения

$$u_j^i = V^i(1 - nGr^i) - V_j\left(1 - mGr_j\right) + \\ + Gr^i \sum_{j=1}^{n} V_j - Gr_j \sum_{i=1}^{m} V^i. \quad (21)$$

Матрица $[u^i_j]$ с элементами из (21) состоит из двух частей: матрицы, соответствующей прямому подключению, (см. (19)) и матрицы, линейно зависящей от сопротивлений резисторов подключения. Именно вторая часть – модуляция – представляет для нас интерес, поскольку она описывает влияние резисторов подключения на распределение напряжений на переменных резисторах. По мере роста сопротивлений резисторов подключения вклад модуляции растёт (по абсолютной величине). Однако большие сопротивления резисторов подключения приводят к большим падениям напряжения на этих резисторах, что сильно ограничивает эффективность модуляции. Наибольшая модуляция достигается при $g^i \simeq nG, g_j \simeq mG$. Для оценки общего характера пространственной модуляции можно пользоваться линейным по r^i, r_j приближением (21).

Рассмотрим конкретную схему записи (одну из возможных), демонстрирующую возможности метода управления за счёт изменения сопротивлений резисторов подключения.

4. Схема записи

Пусть

$$V^i = V^0\delta^{i1}, V_j = V_0\delta_{j1}, \quad (22)$$

то есть гармонические напряжения (с одинаковой частотой) подаются только на первый горизонтальный и первый вертикальный проводники кроссбара, остальные проводники подключены к земле через соответствующие резисторы подключения. С помощью (18) получаем следующее распределение напряжений:

$$u_j^i = \frac{g^1V^0}{g^1+nG}\left[\delta^{i1} + \frac{G}{g^i+nG}\frac{f_0}{1-f^0f_0} - \frac{G}{g_j+mG}\frac{1}{1-f^0f_0}\right] - \\ - \frac{g_1V_0}{g_1+mG}\left[\delta_{j1} + \frac{G}{g_j+mG}\frac{f^0}{1-f^0f_0} - \frac{G}{g^i+nG}\frac{1}{1-f^0f_0}\right]. \quad (23)$$

Более понятные выражения получаются при выполнении условий (20):

$$u_j^i = V^0[\delta^{i1}(1 - r^1nG) - Gr_j] - V_0[\delta_{j1}(1 - \\ - r_1mG) - Gr^i]. \quad (24)$$

Здесь хорошо видно, что вклад, соответствующий прямому подключению, сосредоточен в первой строке и первом столбце матрицы $[u^i_j]$. При условиях (20) этот вклад существенно превышает модуляционный вклад. Поэтому первую строку и первый столбец резисторной матрицы не стоит использовать для записи информации. Более того, чтобы избежать значительных отклонений от базового состояния, желательно в качестве резисторов первой строки и первого столбца использовать постоянные резисторы с проводимостью G.

Для остальных резисторов, используемых для записи информации, получаем совсем простые выражения для комплексных амплитуд напряжения:

$$u_j^i = G(V_0r^i - V^0r_j). \quad (25)$$

Рассмотрим специальные распределения сопротивлений резисторов подключения.

А. При

$$r^i = r^0, r_j = r_0, \quad (26)$$

(одинаковые сопротивления резисторов подключения каждого типа) получаем

$$u_j^i = G(V_0r^0 - V^0r_0), \quad (27)$$

то есть имеем пространственно однородное воздействие – равномерное распределение напряжений. Такое воздействие приводит к одинаковому смещению состояний резисторов матрицы:

$$\Delta x_j^i = \Delta x(|u_j^i|) = \Delta x(G|V_0r^0 - V^0r_0|) \quad (28)$$

Величину смещения можно регулировать, меняя время записи T, амплитуды (вещественные) напряжений источников $|V^0|$, $|V_0|$, разность фаз между напряжениями источников, или сопротивления r^0, r_0. Отметим, что смещение зависит от положительной вещественной амплитуды напряжения, что несколько ограничивает возможности получения произвольного смещения. Расширить возможности можно при использовании напряжений с нулевой частотой. При этом смещение зависит от самого напряжения, которое может быть как положительным, так и отрицательным.

Б. При

$$r^i = r^0\delta^{ip}, r_j = r_0\delta_{jq}, \quad (29)$$

(сопротивление только одного резистора подключения из каждого набора отлично от нуля)

получаем из (25)

$$u_j^i = G(V_0 r^0 \delta^{ip} - V^0 r_0 \delta_{jq}). \quad (30)$$

Ненулевое воздействие осуществляется на резисторы p-ой строки и q-го столбца резисторной матрицы, причем степень воздействия (вещественная амплитуда напряжения) на эти резисторы

$$|u_{j\neq q}^p| = Gr^0|V_o|,$$
$$|u_q^{i\neq p}| = Gr_0|V^o|, \quad .(31)$$
$$|u_q^p| = G|V_0 r^0 - V^o r_0|,$$

Путем выбора управляемых параметров можно получить полезные распределения воздействий. Так, при

$$V_0 r^0 = V^0 r_0 \exp(\pm i\,\pi/6). \quad (32)$$

получаем одинаковые воздействия для всей резисторов p-ой строки и q-го столбца: $|u_{j\neq q}^p| = |u_q^{i\neq p}| = |u_q^p| = Gr^0|V_0|$. При

$$V_0 r^0 = -V^0 r_0. \quad (33)$$

согласно (31) получаем $|u_q^p| = 2|u_{j\neq q}^p| = 2|u_q^{i\neq p}| = 2Gr^0|V_0|$ – одинаковое воздействие на резисторы выделенных строки и столбца, но с суммированием воздействий на пересечении строки и столбца. При

$$V_0 r^0 = V^0 r_0. \quad (34)$$

получаем $|u_q^p| = 0, |u_{j\neq q}^p| = |u_q^{i\neq p}| = Gr^0|V_0|$ – одинаковое воздействие на резисторы выделенных строки и столбца, но с вычитанием воздействий на пересечении строки и столбца.

В. Пусть

$$r^i = r^0(1-\delta^{ip}), r_j = r_0(1-\delta_{jq}) \quad (35)$$

(для каждого набора сопротивления всех резисторов подключения кроме одного отличны от нуля и одинаковы). Тогда согласно (25)

$$u_j^i = G(V_0 r^0(1-\delta^{ip}) - V^0 r_0(1-\delta_{jq})). \quad (36)$$

(36)

Только для резистора $R^i{}_j$ имеем нулевое воздействие (в общем случае). Все остальные резисторы подвергаются воздействию, которое можно регулировать, меняя управляющие параметры. В частности, при выполнении условия (32) все резисторы кроме выделенного подвергаются одинаковому воздействию: $|u_{j\neq q}^p| = |u_q^{i\neq p}| = |u_{j\neq q}^{i\neq p}| = Gr^0|V_0|$.

Г. Пусть

$$r^i = r^0(1-\delta^{ip}), r_j = r_0\delta_{jq} \quad (37)$$

(гибрид случаев Б и В). Здесь

$$u_j^i = G(V_0 r^0(1-\delta^{ip}) - V^0 r_0 \delta_{jq}). \quad (38)$$

Нулевое воздействие в общем случае имеет место только для резисторов $R^p{}_j$ при $j\neq q$. Однако при специальном выборе параметров можно получить дополнительные резисторы с нулевым воздействием. При выполнении условия (34) нулевое воздействие испытывают также резисторы $R^i{}_q$, $i\neq p$, а остальные резисторы испытывают одинаковое воздействие: $|u_q^p| = |u_{j\neq q}^{i\neq p}| = Gr^0|V_0|$. Полученное распределение воздействий – дополнительное по отношению к распределению в случае Б при условии (34).

5. Процедура локальной записи

Для локальной (точечной) записи необходимо изменить состояние (проводимость) целевого резистора, при этом состояния остальных резисторов должны остаться неизменными. Рассмотренные в предыдущем разделе варианты схемы записи не обеспечивают локальности – многие нецелевые резисторы претерпевают изменения состояния. Чтобы получить локальную запись требуется комбинация нескольких этапов записи, в результате которой происходит компенсация изменения состояний нецелевых резисторов. Рассмотрим две комбинации такого рода.

Пусть на первом этапе используется вариант В, то есть выбраны сопротивления резисторов подключения согласно (36), причём выполняется условие (32). В результате получаем изменение состояний резисторов

$$\Delta_1 x_j^i = \Delta x(Gr^0|V_0|), (i,j) \neq (p,q), \quad \Delta_1 x_q^p = 0. \quad (39)$$

На втором этапе записи используем вариант А (то есть выбираем сопротивления резисторов согласно (26)) – однородный сдвиг состояний. Параметры выбираем так, чтобы, вернуть состояния нецелевых резисторов в начальное состояние:

$$\Delta_2 x_j^i = -\Delta x(Gr^0|V_0|). \quad (40)$$

После двух этапов получаем изменения состояний резисторов

$$\Delta x_j^i = \Delta_1 x_j^i + \Delta_2 x_j^i = -\Delta x(Gr^0|V_0|)\delta^{ip}\delta_{jq}. (41)$$

В итоге только целевой резистор $R^p{}_q$ испытывает изменение состояния, что и требовалось. Величину изменения можно регулировать, меняя время записи T, сопротивление резисторов подключения r^0 или амплитуду напряжения источника $|V_0|$. Заметим, что из-за наличия знака «-» в правой части уравнения (40) однородный сдвиг состояний удобнее производить с помощью постоянных напряжений (в случае положительности изменения состояния $\Delta x(Gr^0|V_0|)$, что обычно имеет место).

Можно использовать другую процедуру, приводящую к положительному изменению состояния целевого резистора. На первом этапе используем вариант Б (см. (29)) при выполнении условия (33). Получаем изменения состояний

$$\Delta_1 x_q^p = \Delta x(2Gr^0|V_0|),$$
$$\Delta_1 x_j^p = \Delta_1 x_q^i = \Delta x(Gr^0|V_0|), i \neq p, j \neq q.$$
$$\Delta_1 x_j^i = 0, i \neq p, j \neq q.$$
(42)

На втором этапе используем вариант Г схемы записи (сопротивления резисторов подключения (37)) при выполнении условия (34), так что

$$\Delta_2 x_q^p = \Delta_2 x_j^i = \Delta x(Gr^0|V_0|), i \neq p, j \neq q,$$
$$\Delta_2 x_j^p = \Delta_2 x_q^i = 0, i \neq p, j \neq q.$$
(43)|

После двух этапов получаем

$$\Delta x_j^i = \Delta_1 x_j^i + \Delta_2 x_j^i,$$
$$\Delta x_q^p = \Delta x(2Gr^0|V_0|) + \Delta x(Gr^0|V_0|),$$
$$\Delta x_j^i = \Delta x(Gr^0|V_0|), (i,j) \neq (p,q).$$
(44)

Все нецелевые резисторы имеют одинаковое смещение состояния. Чтобы его компенсировать, необходимо применить однородный сдвиг (вариант А). Выбирая

$$\Delta_3 x_j^i = -\Delta x(Gr^0|V_0|), \quad (45)$$

получаем после трёх этапов

$$\Delta x_j^i = \Delta_1 x_j^i + \Delta_2 x_j^i + \Delta_3 x_j^i = \\ = \Delta x(2Gr^0|V_0|)\delta^{ip}\delta_{jq}(46).$$

Опять пришли к точечной записи. В отличие от выражения (41) в формуле (6) нет знака «-». Во многих случаях это можно рассматривать как достоинство [6], что оправдывает наличие трёх этапов вместо двух.

Отметим наличие значительно вырождения по параметрам записи, что позволяет наложить дополнительные ограничения на эти параметры. Например, можно принять, что $r^0=r_0$, $|V^0|=|V_0|$. При этом удовлетворить условиям (32)—(34) можно за счёт выбора разности фаз между гармоническими сигналами двух источников напряжения. Более того, можно зафиксировать величину ненулевых сопротивлений резисторов подключения. Это означает, что вместо схемы, устанавливающей значение сопротивления резистора подключения можно использовать значительно более простую схему переключения между прямым подключением проводника кроссбара к источнику и подключением с помощью фиксированного резистора. Для реализации двух источников гармонического напряжения можно использовать один генератор гармонического сигнала и управляемый фазовращатель, задающий разность фаз между сигналами источников.

6. Заключение

Итак, варьирование сопротивлений резисторов подключения при использовании высокочастотного гармонического сигнала позволяет производить точечную запись информации в резисторную матрицу, то есть **менять проводимость произвольно выбранного переменного резистора** на заданную величину. Это позволяет формировать произвольную матрицу проводимостей (из допустимого диапазона). Особенно полезен рассматриваемый метод, когда требуется произвести изменения проводимостей небольшого числа резисторов. Если же требуется сформировать матрицу проводимостей «с нуля», лучше использовать другой метод (см., например [7]).

Заметим, что в принципе можно пропустить последний этап процедуры записи – однородный сдвиг состояний. Однородное – одинаковое для всех резисторов – смещение состояний можно вычесть математически. Фактически это означает использование переменного базового состояния x_b. Для отслеживания изменений базового состояния необходимо выделить один или несколько (для большей точности) резисторов, в которые не производится запись информации. Эти резисторы находятся в текущем базовом состоянии, которое легко считать. Правда, при большом числе шагов, изменение базового состояния может оказаться большим, что нежелательно из-за ухудшения эффективности записи информации. Тем не менее, использование плавающего базового состояния может оказаться полезным при заметной релаксации резисторов к основному состоянию, так как в этом случае затруднительно поддерживать постоянное базовое состояние.

Удобство и сама возможность использования высокочастотных сигналов для формирования матрицы зависит от вида характеристических функций резистора. В частности, наиболее благоприятные условия возникают, если функция $F_I^+(I)$ растёт быстрее функции $F_I^-(I)$ при увеличении тока I [6]. В этом случае доступен в принципе весь диапазон состояний резистора, и легко получить как положительные, так и отрицательные значения Δx. В противном случае доступный диапазон состояния уже, меньше возможностей для выбора базового состояния, время записи увеличивается, поскольку нельзя использовать большие амплитуды тока для получения положительных смещений Δx. Для функций $F_0(x), F_x^+(x), F_x^-(x)$ мы приняли выполнение условий (4). Невыполнение этих условий может сделать невозможной запись с помощью высокочастотных сигналов. Впрочем, условия (4) кажутся вполне естественными и даже имеют экспериментальные подтверждения [9].

При записи с помощью высокочастотных напряжений направление резистора не имеет значения. Возникает желание использовать

ненаправленные или двунаправленные переменные резисторы, для которых противоположные направления тока могут приводить к одинаковым знакам изменения проводимости резистора. Это вполне возможно, однако происходящие при записи процессы имеют специфические особенности. Например, при использовании двунаправленного переменного резистора, эквивалентного двум антипараллельным простым резисторным элементам [10], возникает усложнение, связанное с наличием двух переменных состояния. В этом случае величина сопротивления не определяет однозначно переменные состояния. Одинаковые воздействия при одинаковых начальных сопротивлениях могут приводить к разным результатам – изменениям сопротивления резистора. Вообще наличие дополнительных скрытых переменных состояния резистора сильно затрудняет получение предсказуемых результатов воздействия (приложения напряжения). Повышенная сложность элементов может мешать решению простых задач типа записи информации или формирования матрицы для вектор-матричного умножения.

Работа выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН по теме № FNEF-2024-0001 "Создание и реализация доверенных систем искусственного интеллекта, основанных на новых математических и алгоритмических методах, моделях быстрых вычислений, реализуемых на отечественных вычислительных системах" (1023032100070-3-1.2.1).

Local Point Recording of Information into a Crossbar Resistor Array

V.B. Kotov[1], G.A. Beskhlebnova[2]

Abstract. For a large crossbar resistor array, local recording of information into the selected variable resistor (that is, changing the conductivity of this resistor) faces difficulties associated with a limited number of control signals - voltages on the conductors of the structure. Since the number of conductors is significantly less than the number of resistors, when voltage is applied to the target resistor, voltages arise on many non-target resistors. The corresponding changes in the conductivities of non-target resistors must be compensated. The paper considers the recording procedure using high-frequency harmonic signals and with addressing control by varying the resistances of the connection resistors. Based on the analysis using the model of a simple resistor element, the possibility of point recording of information into a resistor array, that is a local change in the conductivity matrix, is shown. The conditions ensuring the feasibility and convenience of such a procedure are discussed.

Keywords: variable resistor, high-frequency signal, crossbar resistor array, conductivity matrix, point recording

Литература

1. 2015.Adamatzky A., Chua L. Memristor Networks. Springer International Publishing (2014).
2. Advances in Memristors, Memristive Devices and Systems. /Edited by S. Vaidyanathan and C. Volos. Springer In-ternational Publishing AG (2017).
3. Kim S. Ju, Kim S., Jang H.W. Competing memristors for brain-inspired computing. iScience 24, 101889, Janu-ary 22, 2021.
4. Kotov V.B., Beskhlebnova G.A. Specifics of Crossbar Resistor Arrays. //B. Kryzhanovsky et al. (Eds.). Ad-vances in Neural Computation, Machine Learning, and Cognitive Research VI (NEUROINFORMATICS 2022). Studies in Computational Intelligence. Vol. 1064. Cham: Springer. 2023. PP. 292–304. https://doi.org/10.1007/978-3-031-19032-2_31.
5. Kotov V.B., Beskhlebnova G.A. Generation of the Conductivity Matrix. //B. Kryzhanovsky et al. (Eds.). Advances in Neural Computation, Machine Learning, and Cognitive Research V (NEUROINFORMATICS 2021). Studies in Computational Intelligence. Vol. 1008. Cham: Springer. 2022. PP. 276-284.
6. Beskhlebnova G.A., Kotov V.B. The Variable Resistor Under a High-Frequency Signal. //B. Kryzhanovsky et al. (Eds.). Advances in Neural Computation, Machine Learning, and Cognitive Research VII (NEUROINFORMATICS 2023). Studies in Computational Intelligence. Vol. 1120. Springer Nature Switzerland AG. 2023. PP. 257–266. https://doi.org/10.1007/978-3-031-44865-2_28.
7. Kotov V.B., Beskhlebnova G.A. Use of High-Frequency Signals to Generate a Conductivity Matrix

//B. Kryzhanovsky et al. (Eds.). Advances in Neural Computation, Machine Learning, and Cognitive Research VIII (NEU-ROINFORMATICS 2024). Studies in Computational Intelligence. Cham: Springer. 2024 (to be published).

8. Kotov V.B., Yudkin F.A. Modeling and Characterization of Resistor Elements for Neuromorphic Systems. Op-tical Memory and Neural Networks (Information Optics). 2019, v.28, No.4, P. 271-282.

9. Surazhevsky I.A. at all. Noise-assisted persistence and recovery of memory state in a memristive spiking neuromorphic network.Chaos, Solitons and Fractals. 146 (2021). 110890.

10. Kotov V.B., Pushkareva M.M. The Bidirectional Variable Resistor Model. //B. Kryzhanovsky et al. (Eds.). Ad-vances in Neural Computation, Machine Learning, and Cognitive Research V (NEUROINFORMATICS 2021). Studies in Computational Intelligence. Vol. 1008. Cham: Springer. 2022. P. 177-186.

Применение функции Ламберта для моделирования ВАХ GAA нанотранзисторов

Н.В. Масальский[1]

[1]НИЦ «Курчатовский институт» – НИИСИ, Москва, Россия, volkov@niisi.ras

Аннотация. Исследуется возможность применения специальной функции Ламберта для моделирования ВАХ кремниевых полевых транзисторов. Разработана аналитическая модель кремниевого полевого GAA нанотранзистора с цилиндрической геометрией рабочей области. При этом в модели транзистора, сформулированной в рамках зарядового разделения, интегральное выражение для тока транзистора заменено аналитическим с использованием функции Ламберта. Результаты расчетов ВАХ сопоставляются с результатами моделирования полученными при помощи широко используемой среды программно- технологического моделирования. Финальная формулировка модели характеризуется следующими преимуществами: она является аналитической, адекватной и компактной. Достигается высокая точность при минимальных вычислительных затратах. Это позволяет использовать рассмотренный подход в инструментах проектирования и поиске начального приближения для трех мерного приборно-технологического моделирования.

Ключевые слова: кремниевый нанотранзистор с полностью охватывающим затвором, функция Ламберта, аналитическая модель, вольт-амперные характеристики, моделирование

1. Введение

Стремительное развитие технологий микроэлектроники открывает возможности разработки различных транзисторных архитектур [1, 2]. Это обусловлено конфликтующими требованиями к микросхемам для различных приложений. Например, в дорожной карте [2] выделено три семейства транзисторных структур. Соответственно, требуются модели транзисторов для каждого уровня разработки. Аналитические модели важны на каждом из них. Поэтому их эффективность определяется, с одной стороны, низким уровнем вычислительных затрат, с другой – адекватностью описания ВАХ транзистора. Немаловажной характеристикой такой модели является возможность калибровки и извлечения параметров для схемотехнического моделирования [3]. Усложнение конструкций привело к тому, что чисто аналитические модели трансформировались в квази аналитические. В частности, ток транзистора (ток стока) вычисляется уже при помощи интегрирования распределения заряда в рабочей области (канала) транзистора. Очевидно, что это увеличивает время расчетов [4, 5]. При этом, большая часть характеристик, например, распределение потенциала, вычисляется с помощью аналитических выражений. Одним из способов вернуться к полностью аналитической модели является замена выражения для тока транзистора аналитическим «двойником», полученным, в частности, с использованием специальных функций, например, функции Ламберта [6]. Специальная функция Ламберта W_L использовалась рядом авторов, начиная с Эйлера, при решении различных математических проблем. Эта функция дифференцируема, интегрируема. В системе компьютерной алгебры MAPLE построены эффективные процедуры вычисления ее значений.

Функция Ламберта $W_L(x)$ задается в неявном виде, как решение уравнения $W_L(x)\exp(W_L(x)) = x$ [6]. Существуют две ветви множества решений – комплексная и действительная. В дальнейшем нас будет интересовать только действительная область решений. В области действительных решений выделяют две ветви - главную $W_{L(0)}(x)$ и вторую $W_{L(-1)}(x)$ -При этом вторая является продолжением главной.

Архитектура GAA (gate all-around или с полностью охватывающим затвором) в настоящее время занимает ведущие позиции в современных разработках нанотранзисторных цифровых СБИС [1, 2]. Цилиндрическая геометрия GAA транзистора (его упрощенная схема приведена ниже на рис. 1) является наиболее универсальным отражением реальных устройств, поскольку в процессе изготовления, ребра отдельных областей транзистора, как правило, скругляются [3, 7]. Сложность модели цилиндрического GAA состоит в том, что необходимо решить 3D уравнение Пуассона в цилиндрические координаты, что является гораздо более сложной задачей, чем в декартовых координатах [5, 8]. Цель данной работы оценить эффективность моделирования ВАХ при помощи функции Ламберта на примере кремниевого цилиндрического GAA транзистора. При этом в модели транзистора, сформулированной в рамках зарядового разделе-

ния, интегральное выражение для тока транзистора заменено аналитическим с использованием функции Ламберта. Первоначально, мы рассматриваем длинно-канальный (субмикронный) прототип транзистора, дабы избежать влияния коротко-канальных эффектов. Следует отметить, что решение уравнения Пуассона получено на основе параболического аппроксимирующего распределения потенциала [7, 9].

2. Аналитическая формулировка модели

Рассмотрим переход от численной к аналитической формулировке модели. Очевидно, что будет присутствовать определенная потеря точности из-за сделанных допущений. Однако, и прогнозируется уменьшение вычислительных затрат. Используя выражение для поверхностного потенциала, полученное исходя из теоремы Гаусс и граничных условий, основное зарядовое уравнение относительно подвижного заряда Qm можно представить в виде:

$$Q_m \exp(Q_m / C_{ox}\varphi_T) = Q_S\, Q_A \times \times \frac{1-\exp(-Q_t / Q_S)}{Q_t} \exp(\gamma)) \qquad (1)$$

где $Q_S = 2\varepsilon_S \varphi_T / R$, $Q_A = qN_A R/2$, $Q_t = Q_S - Q_A$,

$\gamma = (U_{gs} - U_{FB} + \frac{Q_A}{C_{ox}} - U)/\varphi_T$, N_A – концентрация легирующей примеси рабочей области, Cox - емкость подзатворного диэлектрика, Ugs- напряжение на затворе, U_{FB} - напряжение плоских зон, φ_T - термический потенциал. Его решение может быть получено с помощью функции Ламберта W_L. Отправной точкой является приближение для режима сильной инверсии [9, 10]. В данном случае ограничимся только квадратичным членом. Тогда приемлемую аппроксимацию распределение заряда в рабочей области транзистора во всех режимах работы аналитически можно поучить из решения (1) следующим образом:

$$Q_m^s = 2C_{ox}\varphi_T W_L\left(\frac{Q_S Q_A}{C_{ox}\varphi_T} \times \times \frac{1-\exp(\frac{Q_A - Q_m^w}{Q_S})}{Q_m^{'} - Q_A} \exp(\gamma)\right) \qquad (2)$$

$$Q_m^w = 2C_{ox}\varphi_T W_L\left(\frac{Q_S Q_A}{2C_{ox}\varphi_T} \times \times \frac{\exp(\frac{Q^{'}}{Q_S}) - \exp(\frac{Q_A}{Q_S})}{Q^{'} - Q_A} \exp(\gamma)\right) \qquad (3)$$

где $Q_m^{'} \approx 2C_{ox}\varphi_T W_L\left(\frac{\sqrt{Q_S Q_A}}{2C_{ox}\varphi_T} \exp(\gamma)\right)$.

Следовательно, и (2), и (3) являются приближенными решениями (1). Но выражение (3) более точно описывает распределение заряда при обеднении, тогда как (2) лучше подходит для режима аккумуляции. Поэтому (2) и (3) в некотором роде являются самосогласованными и которые справедливы в обоих режимах работы транзистора. Следует отметить, что, т. к. общее решение достигается методом итерации между (2) и (3), переход между режимами обеднения и аккумуляции достигается естественным образом, что позволяет не использовать «сглаживающих функций» [9]. Дополнительное повторение расчетов даже по одному циклу позволяет повысить точность результатов [11]. Далее, можно усовершенствовать модель – используя приближение для функции Ламберта [12] с минимальной погрешностью

$W_L(x) = \ln(1+x)\left(1 - \frac{\ln(1+\ln(1+x))}{2+\ln(1+x)}\right)$ что дополнительно увеличит ее эффективность.

Аналитическое выражение для тока стока Ids транзистора получено следующим образом. Дифференцируя (1), определяем зависимость $d\gamma(dQ)$. Следующий шаг - интегрирование общего выражения для тока (см, например, [13]) по dQ. В данном случае точное решение включает специальные функции - полилогарифмы и плохо сходящиеся ряды [14]. Их наличие затрудняет численную оценку [11]. Данная проблема была решена заменой этих функций на приближенные, которые являются непрерывными, дифференцируемыми и интегрируемыми. В результате этой замены погрешность расчетов увеличилась в окрестности точки перехода режимов работы транзистора. Окончательное выражение для тока стока можно записать в виде:

$$I_{ds} = 2\pi\varphi_T \frac{R}{L_g} \mu_{eff} \times \times (f(Q_m(0)) - f(Q_m(U_{ds})))$$

$$f(Q) = (2 + \frac{Q}{2C_{ox}\varphi_T} -$$

где $-\beta \ln(1 + \exp(\frac{Q - Q_A}{2\beta Q_S})))Q +$ и

$$+Q_A \ln(\frac{Q - Q_A}{2Q_S} \frac{1}{\exp(\frac{Q - Q_A}{2Q_S}) - 1})$$

μ_{eff} - эффективная подвижность электронов, подчиняющаяся общепринятому правилу Маттессена, β – безразмерный коэффициент зарядового разделения, в некотором смысле - подгоночный параметр.

В данном представлении наибольшая ошибка связана с напряжением плоских зон и синхронно возрастает с ростом отношения Q_A / Q_S. Следует отметить, что транзисторы высокими значениями Q_A / Q_S являются редко используемые в практических приложениях, из-за порогового напряжение близкого к нулю [3, 15]. Следует отметить, что разработанная аналитическая модель тока стока может быть применима для других конструкций кремниевых цилиндрических КМОП транзисторов с полностью охватывающим затвором, поскольку модель плотности подвижного заряда зависит только от значений этой плотности у истока Qm (0) и у стока Qm (Uds) транзистора, где Uds – напряжение на стоке транзистора (исток заземлен). А выражения для этих и величин получены из (2) и (3) при U = 0 и U = Uds, соответственно.

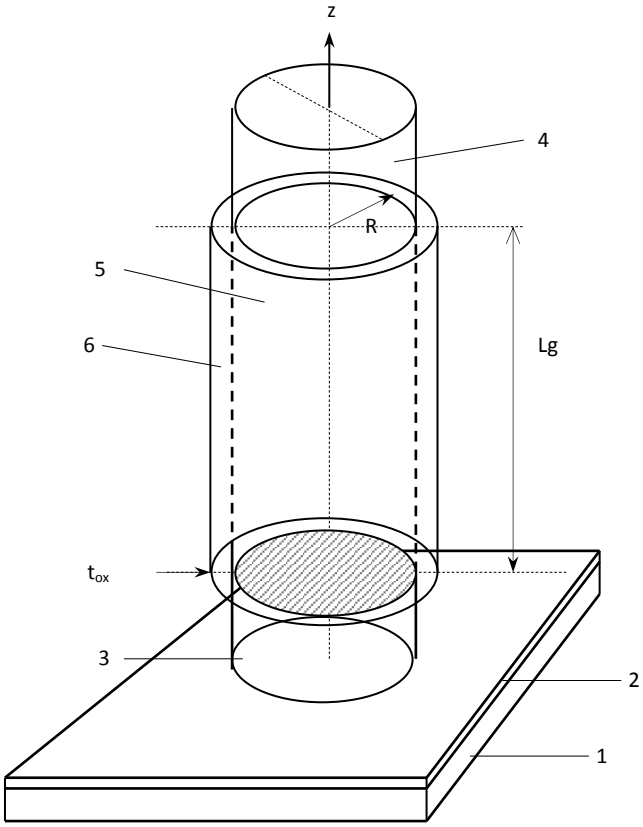

Рис. 1. Упрощенная схема кремниевого вертикального транзистора цилиндрической геометрии с полностью охватывающим затвором, где 1 – подложка, 2 – пленка оксида кремния, 3- исток, 4 - сток, 5- рабочая область, 6 – подзатворный диэлектрик (тело затвора не показано), Lg, R – длина затвора и радиус рабочей области, tox – толщина подзатворного окисла, ось z, начало координат на верхней грани истока.

3. Результаты сравнительного моделирования

Для численных экспериментов выбран следующий прототип кремниевого цилиндрического КМОП транзистора с полностью охватывающим затвором. Для данной модели длина рабочей области Lg фиксирована и составляет 0.35 мкм. Исходное значение радиус рабочей области R 5 нм. Концентрация легирования N_A равна 1.5×10^{18} см$^{-3}$. Исток и сток равномерно легированы примесью с концентрацией $N_{SD} = 0.5 \times 10^{20}$ см$^{-3}$. Затвор поликремния был выбран для того, чтобы получить максимально возможное пороговое напряжение (Uth). Толщина затвора равна 20 нм. Толщина подзатворного оксида кремния tox составляет 2.5 нм. Работа выхода металлического электрода затвора составляет 4.6 эВ.

Модель была проверена с помощью широко применяемого мирового стандарта приборно-технологического моделирования. В обоих моделях использовались те же значения физических констант [13], чтобы уменьшить рассогласования в описании физических процессов переноса. В разработанной модели не учитывается форма истока/стока – они считаются кубическими с размерами, значительно превышающими значение R. При этом считается, что границы всех областей транзистора являются идеальными. Чтобы максимально снизить последовательное сопротивление исток-рабочая область–сток, мы увеличили уровень их легирования. При этом мы не рассматриваем любые механизмы туннелирования в обоих моделях.

На рис. 2 представлены результаты моделирования двумя методами.

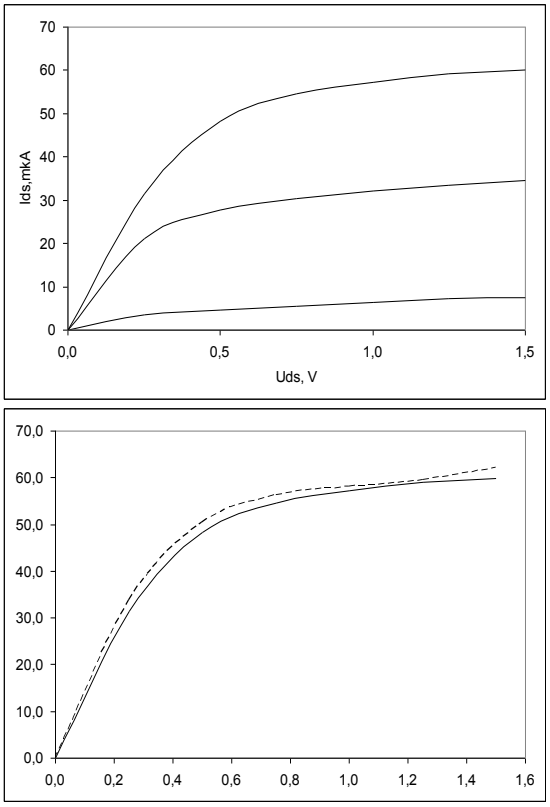

Рис. 2. ВАХ тестового прототипа: верхний рис. - Ids(Uds) при Ugs = 0.5; 1.0; 1.5 В; средний рис. – Ids(Uda) при Ugs=1.5 В, где сплошная кривая – модель, пунктирная – TCAD-модель; нижний рис.- ВАХ Ids(Ugs) при Uds=0.05 и 1.5

Общая оценка соответствия результатов расчетов для выбранной структуры – достигается вполне приемлемая точность. Очень хорошее совпадение в подпороговом режиме (менее 1%) и режиме насыщения (менее 3%). Немного хуже (чуть более 5%) при переходе между режимами (начальная часть на линейного участка) и при высоких значениях Uds и Ugs. Рис. 3 иллюстрирует влияние изменения радиуса R и толщины tox на параметр Uth в нашей модели.

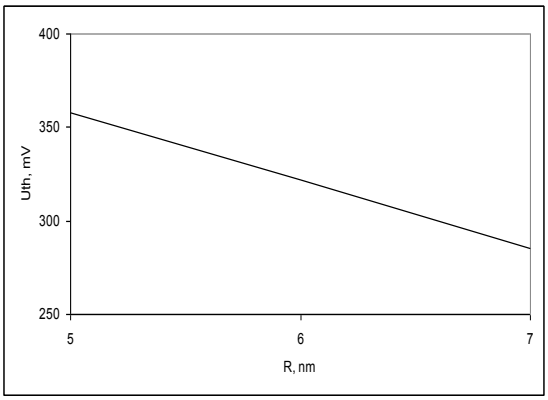

Рис. 3. Зависимость Uth(R) при Lg=0.35 мкм и tox=2.5 нм

Из результатов приведенных на рис. 3 также следует, что модель качественно отражает тенденцию снижения значения Uth с ростом R. Отметим, что зависимость Uth(tox) также отражает вышеуказанную тенденцию аналогично Uth(R).

Ниже на рис. 4 приведены зависимости отклонения значении модельного Uth и рассчитанного при помощи программной среды приборно-технологического моделирования при варьировании параметров R и tox.

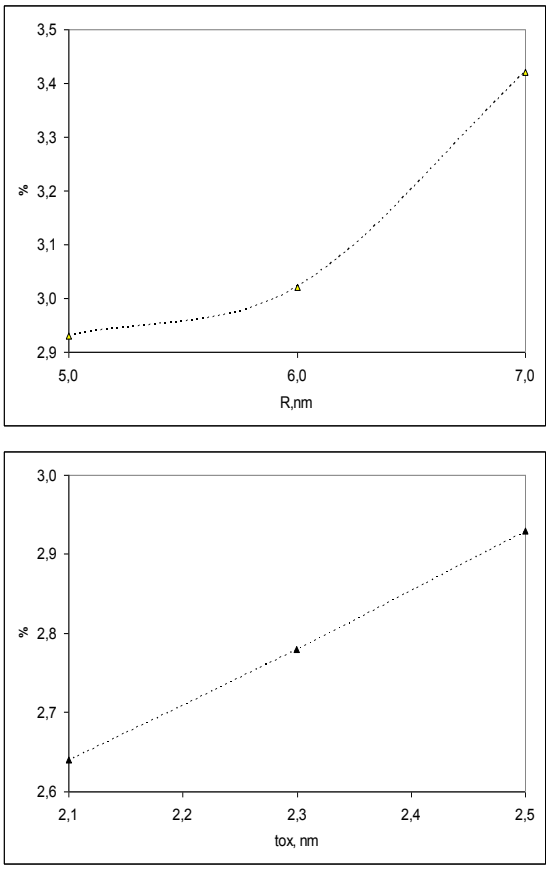

Рис. 4. Зависимость ошибки (в %): верхний рис. - от R при Lg=0.35 мкм и tox=2.5 нм , нижний рис. – от tox R при Lg=0.35 мкм и R=5 нм

В целом, совпадение довольно хорошее, но уровень рассогласования результатов немного (примерно на 2%) увеличивается при высоких значениях R, что особенно заметно в переходной области между подпороговым режимом и выше порога. Напротив, при снижении параметра tox точность увеличивается. Такое поведение ошибки можно объяснить слабым незначительным проявлением коротко-канальных эффектов. Справочно, можно указать, что точность снижается, но будет еще приемлемой (около 20%) при более высоких концентрациях N_A. Однако, этот результат несущественен, т. к. масштабирование в наноразмерной области требует снижения концентрации легирования.

Из анализа результатов, приведенных на рисунках выше, можно сделать вывод, что ВАХ транзистора в нашей модели зависят по ранжиру от N_A, R и затем только от tox. К сожалению, из (1-3) не удалось получить аналитической оценки влияния на ВАХ каждого параметра. Например, изменение R на 1 нм вызывает сдвиг Uth, который примерно в два раза больше, чем аналогичное изменение tox. Однако сдвиг Uth более чувствителен к значению tox, и при ее уменьшении величина сдвига возрастает. Окончательно можно сделать вывод, что необходимо провести более обширные численные эксперименты для проверки адекватности аналитической модели.

4. Заключение

Исследован подход с использованием специальной функции Ламберта для разработки аналитической модели КМОП нанотранзистора. В качестве прототипа выбран кремниевый GAA транзистор с цилиндрической геометрией рабочей области. Финальная формулировка модели характеризуется следующими преимуществами: она является аналитической, адекватной и компактной. Достигается высокая точность при минимальных вычислительных затратах. Это позволяет использовать модель в инструментах проектирования и поиске начального приближения для трехмерного приборно-технологического моделирования.

Публикация выполнена в рамках НИР ФГУ ФНЦ НИИСИ РАН по теме № FNEF-2024-0003 "Методы разработки аппаратно-программных платформ на основе защищенных и устойчивых к сбоям систем на кристалле и сопроцессоров искусственного интеллекта и обработки сигналов".

Application of Lambert Function to Modeling the BAX GAA Nanotransistors

N. Masalsky

Abstract. The possibility of using a special Lambert function to simulate the volt-ampere characteristics of silicon field-effect transistors is investigated. An analytical model of a silicon field-effect GAA nanotransistor with a cylindrical geometry of the working area has been developed. At the same time, in the transistor model formulated within the framework of charge separation, the integral expression for the transistor current is replaced by an analytical one using the Lambert function. The calculation results are compared with the simulation results obtained using a widely used software and technology modeling environment. The final formulation of the model is characterized by

the following advantages: it is analytical, adequate and compact. High accuracy is achieved with minimal computational cost. This makes it possible to use the considered approach in design tools and the search for an initial approximation for three-dimensional applied technological modeling.

Keywords: silicon nanotransistor with fully enveloping gate, Lambert function, analytical model, I-V data, simulation

Литература

1. Г.Я. Красников, Е.С. Горнев, И.В. Матюшкин. Общая теория технологий и микроэлектроника, Техносфера, М, 2020.

2. More Moore. International Roadmap for Devices and Systems. IRDS, Piscataway, NJ, USA, 2021.

3. Г.Я. Красников. Конструктивно-технологические особенности субмикронных МОП транзисторов. Акционерное общество «Рекламно-издательский центр ТЕХНОСФЕРА», 2011.

4. N Sano, K. Yoshida, G. Park. Fundamental aspect of semiconductor device modeling associated with discrete impurities: drift-diffusion scheme. “IEEE Trans. Electron Devices”, (2020), vol. 67, 3323-3328.

5. Н.В. Масальский. Моделирование ВАХ ультратонких КНИ КМОП нанотранзисторов с полностью охватывающим затвором. “Микроэлектроника”, (2021), т. 50, 436-444.

6. R.M. Corless, G.H. Gonnet, D.E.G. Hare, D.J. Jeffrey, D.E. Knuth. On the Lambert W function. «Advances in computational mathematics», (1996), v. 5, 329-359.

7. J.-P. Colinge, FinFETs and Other. Verlag, New York, NY, USA, 2008.

8. G. Tomar, A. Barwari. Fundamental of electronic devices and circuits. Springer, 2019.

9.. M. Lundstrom, J. Guo. Nanoscale Transistors: Device Physics, Modeling and Simulation. Springer: New York, 2006.

10. B. D. Gaynor, S. Hassoun. Fin shape impact on FinFET leakage with application to multithreshold and ultralow-leakage FinFET design. “IEEE Trans. Electron Devices”, (2014), vol. 61, 2738–2744.

11. К.К Абгарян, Д.Л. Ревизников, А.А. Журавлёв, А.Ю. Морозов, Е.С. Гаврилов. Многомасштабное моделирование нейроморфных систем. М.: Изд-во МАКС Пресс, 2022.

12. Ю. Люк. Специальные математические функции и их аппроксимации. М.: Мир, 1980.

13. M. V. Fischetti, W. G. Vandenberghe. Advanced Physics of Electron Transport in Semiconductors and Nanostructures, New York, U.S.A.: Springer, 2016.

14. S. Winitzki. Uniform Approximations for Transcendental Functions. Berlin, Germany: Springer-Verlag, 2003.

15. F. Lime, R. Ritzenthaler, M. Ricoma, F. Martinez, F. Pascal, E. Miranda, O. Faynot, B. Iñiguez. A physical compact DC drain current model for long-channel undoped ultra-thin body (UTB) SOI and asymmetric double-gate (DG) MOSFETs with independent gate operation, “Solid-State Electron.”, (2011), vol. 57, no. 1, 61–66.

Чувствительность распределения потенциала в канале кремниевых GAA нанотранзисторов к аномальному поведению зернистости металлического затвора

Н.В. Масальский[1]

[1]НИЦ «Курчатовский институт» – НИИСИ, Москва, Россия, volkov@niisi.ras.ru

Аннотация. Обсуждается влияние аномальной зернистости металлического затвора кремниевого полевого GAA нанотранзистора с цилиндрической геометрией на распределение потенциала в его рабочей области. На основе аналитического решения 2D уравнения Пуассона разработана аналитическая модель для анализа чувствительности распределения потенциала кремниевых полевых GAA нанотранзисторов к аномальному поведению зернистости металлического затвора. Количественно проанализированы вариации распределения потенциала в транзисторах с короткой и тонкой рабочей областью с длиной от 25 до 11 нм. Показана зависимость возмущения потенциала от расположения аномальной зернистости на затворе. Установлена линейная зависимость амплитуды возмущения от величины скачка работы выхода. Разработана математическая модель флуктуации распределения потенциала, включающая вариации неравномерности границ аномальной области зернистости. Неравномерность границ аномальной области вносит дополнительный вклад в трансформацию потенциала. При примерно одинаковых деформациях границ вклад данного механизма не существенен. При значительной асимметрии границ вклад может превышать 10% от возмущения, сгенерированного идеальным кольцом.

Ключевые слова: кремниевый цилиндрический нанотранзистор с полностью охватывающим затвором, аномальная зернистость металлического затвора, распределение потенциала, уравнение Пуассона, моделирование

1. Введение

Одним из важных источников вариативности электро-физических характеристик кремниевых полевых all around gate (GAA) нанотранзисторов является зернистость металлического затвора (ЗМЗ) [1], возникающая в результате технологических процессов осаждения и литографических процессов [1-3]. В общем случае независимо от материала затвора (или он создан на основе поликристаллического кремния или на металлической основе) затвор состоит из множества хаотически ориентированных зерен с различной конфигурацией [1, 4, 5]. Это приводит к изменению работы выхода затвора φM. В данной работе в качестве объекта исследования выбран широко используемый нитрид-титановый металлический затвор, который характеризуется хорошей совместимостью с оксидом кремния и низкой емкостью [6]. Нивелировать влияние зернистости металлического затвора (ЗМЗ) можно технологическими приемами, снижающими и размер зерна и неравномерность кромки затвора. Использование таких подходов для получения затворов с мелкими зернами может приводить к их аномальному (отличному от равномерного) распределению. Предельным случаем является распределение зерен с одинаковой работой выхода в виде ленты, полностью охватывающей затвор, наблюдаемое экспериментально с помощью KPFM [7]. Следует отметить, что мы не анализируем природу распределения зерен по поверхности затвора и их кластеризацию в определенные домены.

На рис. 1 показана схема кремниевого полевого GAA нанотранзистора с цилиндрической геометрией и его металлического затвора с аномальной зернистостью, на котором изучается влияние механизма анимальной ЗМЗ.

Распределение потенциала - ключевая характеристика любого кремниевого полевого транзистора, включая и GAA архитектуру. В настоящей работе численно исследуется чувствительность распределения потенциала к аномальному поведению зернистости металлического затвора в виде полностью охватывающей области (кольца) и неравномерности ее границы. Исходя из современных технологических норм в настоящей работе диапазон длин затворов (Lg) прототипов составляет от 11 до 25 нм и ширин кольца, соиз-

меримых с характеристической длиной. Влияние эффекта ЗМЗ моделируется путем изменения рабочей функции затвора устройства таким образом, чтобы она соответствовала распределению зерен металла. Из результатов [2-4, 8 9] интересующие нас электро-физические характеристики металлических зерен затвора были выбраны так, что присутствуют только две возможные ориентации металлических зерен с разной работой выхода. Влияние механизма нерегулярности кромки кольца моделируется соответствии с формой заданного профиля шероховатости [2, 36]. Мы рассмотрели два вида неравномерности: плавный (синусоидальный) и резкий (пилообразный) с несколькими значениями периода. При этом считали, что в длину окружности поперечного сечения рабочей области (РО) укладывается целое число периодов.

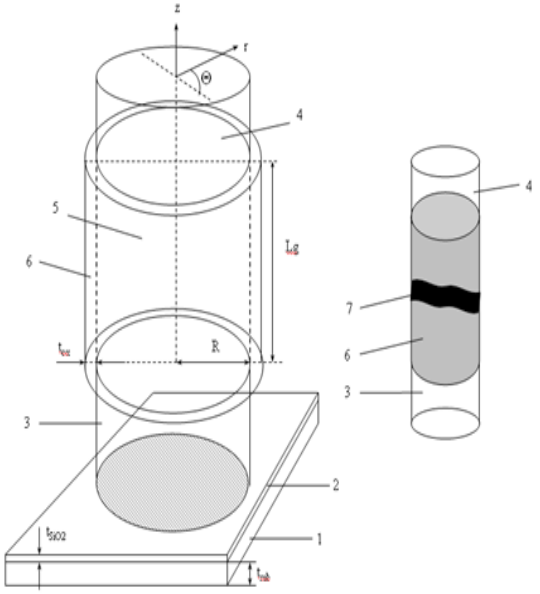

Рис. 1. Схема GAA нанотранзистора с цилиндрической рабочей областью и аномальным распределением зернистости затвора, где 1 – кремниевая подложка, 2 – пленка оксида кремния, 3- исток, 4 - сток, 5- РО, 6 – подзатворный диэлектрик, 7 – затвор с анимальным распределением зернистости – по всей поверхности работа выхода составляет 4.6 эВ (это значение выбрано для примера, в общем случае работа выхода может принимать значения от 4.17 до 5.2 эВ в зависимости от материала или материалов затвора [1]), за исключением узкой ленты (выделено черным цветом) с работой выхода 4.4 эВ,Lg – длина затвора, R - диаметр РО, tox – толщина подзатворного окисла

2. 3D модель распределения потенциала с учетом аномальной ЗМЗ (идеальный случай)

Традиционно для моделирования распределение потенциала в кремниевой цилиндрической РО GAA нанотранзистора используется 2D уравнение Пуассона вида [10]:

(1)

где $\varphi(r,z))$ - электростатический потенциал в рабочей области, q – заряд электрона, ε_S - диэлектрическая проницаемость РО, NA – концентрация легирования РО.

Решение (1) оси z можно записать аналитически в параболическом приближении, предложенном в [10], при соответствующих граничных условиях. Однако в случае аномального поведения работы выхода в виде кольца с гладкими краями напряжение плоских зон U_{FB} будет зависеть от координаты z, которая будет определять место положения кольца с отличным значением работы выхода. Решение (1) можно записать в виде [6]:

$$\phi(r,z)=\phi_c(z)+ +[\frac{\varepsilon_{ox}}{2R\varepsilon_{Si}}\frac{U'_{gs}(z)-\phi_s(z)}{\ln(1+t_{ox}/R)}]r^2, \quad (2)$$

где $\varphi_c(z)=\varphi(0,z)$ - центральный потенциал, $U'_{gs}(z)=U_{gs}-U_{FB}(z)$, U_{gs} - напряжение на затворе, $U_{FB}(z)$ - напряжение плоских зон, ε_{ox}, t_{ox} -диэлектрическая проницаемость и толщина подзатворного оксида, соответственно, R - радиус РО. При этом $\phi_c(z)=(1+C)\phi_s(z)-CU'_{gs}(z)$, где $C=\frac{\varepsilon_{ox}}{2\varepsilon_{Si}}\frac{1}{\ln(1+t_{ox}/R)}$. Тогда потенциал $\varphi(z,r)$ зависит только от выражения $\varphi_s(z)$, которое можно записать так:

$$\phi_s(z) = \frac{1}{\sinh(\frac{L_g}{l})}[\Phi_1(z)\sinh(\frac{L_g - z}{l}) + \Phi_2(z)\sinh(\frac{z}{l})] - A(z) \quad (3)$$

$$A(z) = A_0(z) - U'_{gs}(z),$$

$$\Phi_1(z) = A_0(z) + U_{bi} - U'_{gs}(z),$$

$$\Phi_2(z) = A_0(z) + U_{bi} + U_{ds} - U'_{gs}(z),$$

$$A_0(z) = \frac{qN_A}{\varepsilon_S}(\frac{1 - C((l/L_g)^2 - 1)}{4C})R^2 - U'_{gs}(z),$$

$l = \frac{R}{2}\sqrt{2\frac{\varepsilon_S}{\varepsilon_{ox}}\ln(1+\frac{t_{ox}}{R})+1}$ - характеристическая длина, Uds - напряжение на стоке, Ubi - контактная разность потенциалов.

Такой подход упрощает определение оценки чувствительности распределения потенциала из-за флуктуации топологии затвора. В этом случае можно воспользоваться следующим выражением [11]:

$$\delta\phi_s = \frac{1}{\sinh(\frac{L_g}{l})}[\delta\Phi_1\sinh(\frac{L_g - z}{l}) + \delta\Phi_2\sinh(\frac{z}{l})] - \delta A \quad (4)$$

Полученные соотношения (3) и (4) будут инвариантны относительно значения длины затвора, поскольку мы подразумеваем, что отношение L_g / l практически постоянно в диапазоне исследуемых длин затворов и составляет чуть меньше 10, чтобы выполнить условие подавления коротко-канальных эффектов.

3. Результаты аналитического моделирования

Рассмотрим поведение поверхностного потенциала φs при наличии узкой ленты (полоски), которая полностью охватывает затвор, разной ширины и разных положениях. С точки зрения поведения классического потенциала можно выделить условно пять областей, что иллюстрируется на рис. 2.

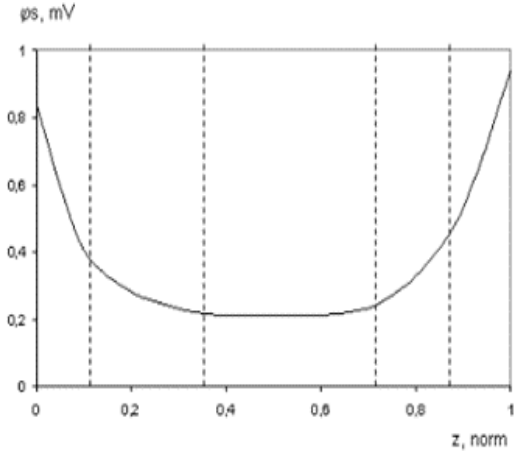

Рис. 2. Распределение потенциала φs в случае φM – const при Uds = 0.1 В

Две быстро меняющиеся, которые расположены рядом со стоком и истоком, две переходные и самую протяженную, в которой потенциал практически не меняется. На рис. 3 кольцо шириной равной характеристической длине располагается практически рядом со стоком, истоком, по середине затвора и в серединах переходных областей от истока и стока.

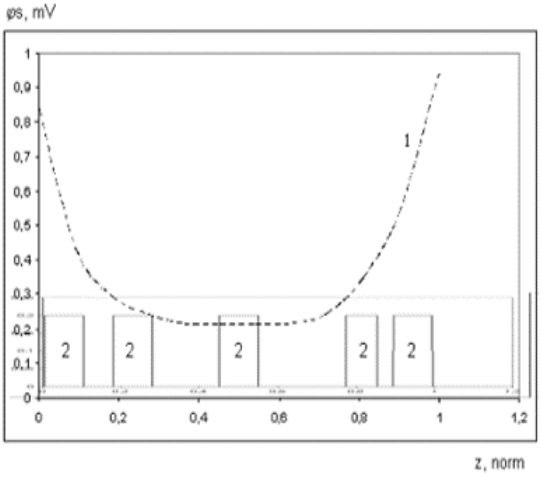

Рис. 3. Расположение скачков работы выхода по длине затвора, где 1 (пунктир) – идеализированное распределение потенциала, 2 – скачки работы выода.

Скачок (амплитуда) возмущения ΔφM ограничена разностью работ выхода из зерен разной ориентации. Следует отметить, что такая схема расположения (по пяти зонам) применима при условии $l \approx G_s$, G_s где размер зерна.

На рис. 4 приведены результаты расчета распределения потенциала в разных областях.

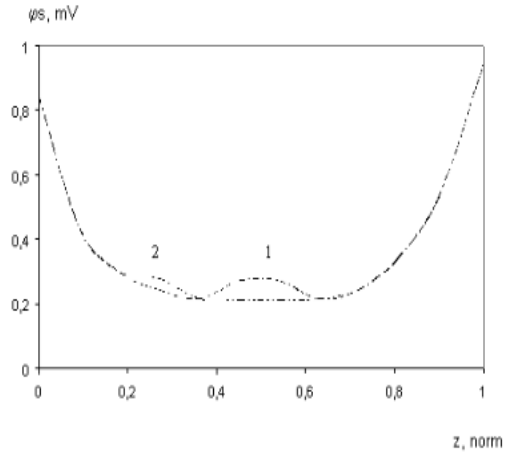

Рис. 4. Распределение потенциала в случае аномального поведения работы выхода

Из приведенных результатов хорошо видно, что отклонение потенциала имеет свои особенности. В областях быстро меняющегося потенциала (с высоким градиентом) их влияние незаметно, в переходных областях - чуть-чуть, и заметное изменение формы распределения потенциала происходит в области, где градиент потенциала близок к 0. Отметим, что для электрофизических характеристик транзистора значение кривизны важно, т. к. оно влияет на ключевые параметры пороговое напряжение и подпороговый наклон.

На рис. 5 приведено распределение потенциала для двух случаев ширины кольца.

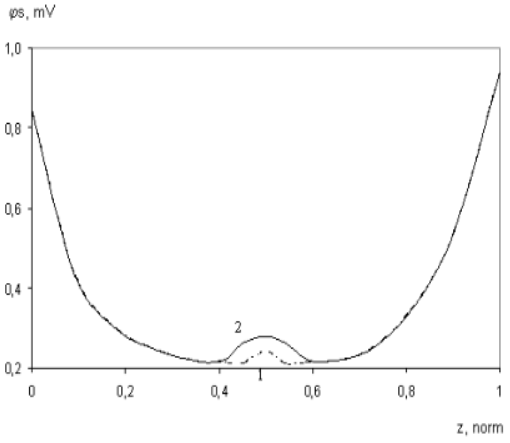

Рис. 5. Распределение потенциала для разной ширины кольца, где 1 - - l/2, 2 - l

Данную зависимость можно обобщить следующим образом. Амплитуда возмущения потенциала Δφs будет пропорциональна ширине кольца. Принимая во внимание, что минимальная ширина кольца ограничена минимальным размером зерна, то для прототипов с Lg<20 нм ширина кольца будет больше чем l. Например, для прототипов с ультра короткой и тонкой РО (случай Lg=11 нм) минимальная ширина кольца будет составлять примерно 2l и конфигурация с 5-ю областями трансформируется в конфигурацию с 3-мя. При этом сохраняется характер зависимости амплитуды возмущения потенциала от ширины и местоположения кольца.

В общем случае амплитуда возмущения потенциала зависит от отношения размера области с отличным значением работы выхода и общей длина затвора Lg. С ростом этого отношения амплитуда возмущения потенциала пропорционально увеличивается. При увеличении ширины кольца линейный характер зависимости сохраняется. Данные зависимости приведены ниже на рис. 6. Следует отметить, что такое масштабирование приведет к замедлению роста амплитуды возмущения потенциала, и в конечном итоге он перестает расти, когда ширина кольца приблизится к половине длины затвора.

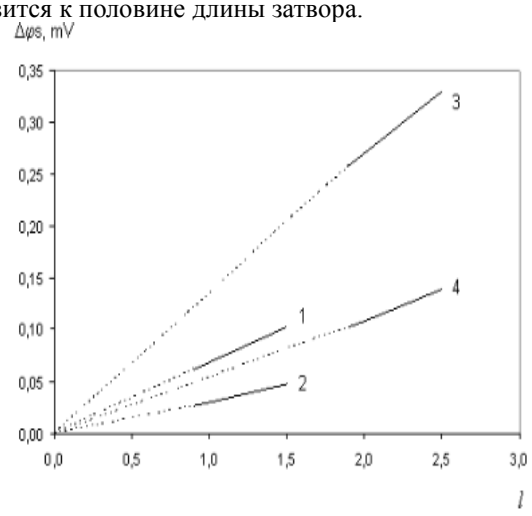

Рис. 6. Зависимость амплитуды возмущения потенциала Δφs от ширины кольца в единицах l, для разных зон и Lg, где 1,2 –Lg=25 нм, 3,4 - Lg=11 нм, 1,3 – для центральной зоны, 2,4 – для промежуточной

Отметим, что продолжения приведенных зависимостей проходят через начало координат. А их наклон кроме перечисленных факторов зависит также и от значения ΔφM. Так, с ростом ΔφM крутизна всех зависимостей Δφs(l) пропорционально увеличивается, что иллюстрируется рис. 7.

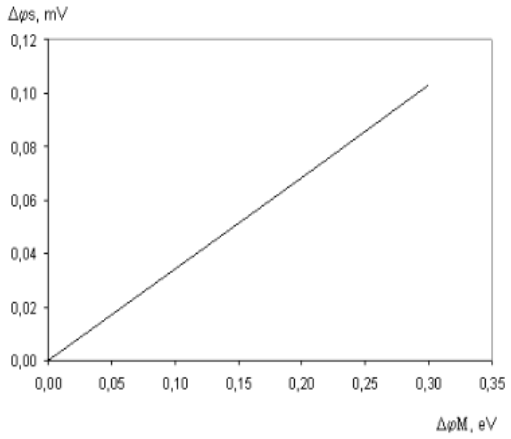

Рис. 7. Зависимость амплитуды возмущения потенциала Δφs (ΔφM)

Наклон этой зависимости будет общим для всех размеров РО в рассматриваемом диапазоне значений при условии, что выполнено условия подавления ККЭ.

Рассмотрим более сложный случай, когда кольцо не перпендикулярно аксиальной оси z, расположено под углом к ней, что иллюстрируется рис . 8.

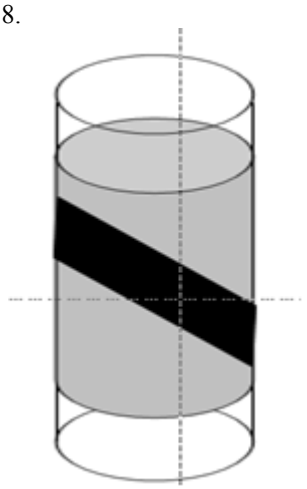

Рис. 8. Схема затвора с наклонной аномалией

В данном случае модель, представленная выше, можно использовать для расчетов распределения потенциала по продольным сечениям с учетом эффективной ширины кольца. В каждом сечении в зависимости от положения области возмущения распределение потенциала будет соответствовать похожим зависимостям, представленными на рис. 4 и 5. Далее следует объединить полученные результаты по угловой координате при фиксированном значении продольной (z) координаты. Альтернативный способ – численно решить 3-х мерное уравнение Пуассона с соответствующими граничными условиями [12]. На рис 9 представлено распределение потенциала для случая для прототипов с ультра короткой РО (Lg=11 нм) с эффективной шириной кольца 2l и углом наклона к аксиальной оси 45 град.

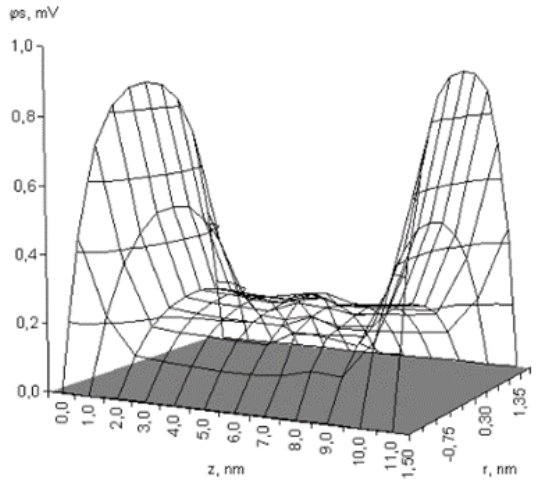

Рис. 9. 3D распределение потенциала

В центральной области амплитуда возмущения в два раза выше, чем в областях, максимально приближенных в истоку и стоку. Если фиксировать эффективную ширину, то характер зависимости распределения потенциала останется неизменным. Этот вывод применим для диапазона углов от 30 до 60 град. При небольших углах наклона до 12 град поведение потенциала практически не отличается от случая, когда кольцо перпендикулярно оси z. Соответственно, амплитуда возмущения ведет себя аналогичным образом - пропорционально изменяется в соответствии с изменением эффективной толщины кольца.

4. Результаты моделирования с учетом неравномерности границы

Шероховатость (неравномерность) границы (кромки) кольца вносит дополнительное изменение амплитуды возмущения потенциала. В общем случае задача моделирования влияния механизма ШГ решается при помощи преобразования Фурье гауссовых спектров [22], [33]. Эти спектры зависят от длины корреляции (Λ) и среднеквадратичного отклонения (Δ). Параметр Λ соотносит распределение деформации границы в соседних точках. Параметр Δ - это высота деформации или отклонение границы от идеальной в направлении вдоль поверхности затвора. Однако, поскольку первоначальная задача отображает маловероятный частный случай, а представление ШГ в виде спектров задача не столько трудоемкая, сколько ресурсозатратная, то мы

рассмотрели два вида неравномерности: плавный (синусоидальный) и резкий (пилообразный) с несколькими значениями периода. При этом считали, что в длину окружности поперечного сечения РО укладывается целое число периодов. А также подразумевалось, что данные профили соответствуют случайным некоррелированным значениям Λ и Δ. Значение параметра Δ во всех случаях составляла 0.5l . Мы анализировали случаи для одного, двух, 5-ти и 10-ти периодов для центральной зоны локализации возмущения работы выхода. При симметричном синусоидальном законе деформации обеих границ не отмечено изменение амплитуды возмущения потенциала при любых положениях кольца вдоль оси z.. Этот факт объясняется тем, что сохраняется «эффективная» (усредненная) ширина кольца. Аналогичный вывод применим, если закон деформации границы кольца соответствует пилообразной функции. Ситуация изменяется, когда границы деформируются со сдвигом в полпериода. Для больших периодов изменения не отмечены. Для малых периодов для случая плавных границ отмечен рост амплитуды возмущения потенциала примерно 0.3% на период, для резких - 0.5% на период. Более существенные изменения в амплитуде проявляются, если законы деформации кромок не совпадают. Например, для случая, когда одна граница синусоидально изменяется за один период, а другая за 10, то амплитуда возмущения потенциала повысилась на 6%. Для случая, если одна граница изменяется по синусоидальному закону с 10-ти кратным периодом, другая «пила» с одним периодом, то рост потенциала на 7%. Если одна граница – однопериодный синус но с удвоенным Δ, другая «пила» с 10-ти кратным периодом, рост на 9%. Поэтому дополнительное смещение потенциала очень существенно зависит от взаимной формы границ. Изменение амплитуды возмущения потенциала более 10% тоже возможны. Например, в случаях большой асимметрии границ. Если одна граница - полупериодный синус, а другая полупериодный «пила», то отмечен рост амплитуды на 12%. Следует отметить, что при большой асимметрии, которая уменьшает «эффективную» ширину кольца, например, в представленном выше случае, только границы вогнуты во внутрь кольца, мы наблюдаем снижение амплитуды возмущения на 11%. Однако, необходимость систематизировать влияние ШГ, по нашему мнению, будет выявлена при исследовании влияния на транспорт носителей и в конечном итоге на ток транзистора и другие его ключевые характеристики.

5. Заключение

На основе аналитического решения 2D уравнения Пуассона разработана аналитическая модель для анализа чувствительности распределения потенциала кремниевых полевых GAA нанотранзисторов к аномальному поведению зернистости металлического затвора. Количественно проанализированы вариации распределения потенциала в транзисторах с короткой и тонкой рабочей областью с длиной от 25 до 11 нм. Показана зависимость возмущения потенциала от расположения аномальной зернистости на затворе. Установлена линейная зависимость амплитуды возмущения от величины скачка работы выхода. Неравномерность границ аномальной области вносит дополнительный вклад в трансформацию потенциала. При примерно одинаковых деформациях границ вклад данного механизма не существенен. При значительной асимметрии границ вклад может превышать 10% от возмущения сгенерированного идеальным кольцом. Подчеркнем, что для кремниевых полевых GAA нанотранзисторов аномальное поведение зернистости металлического затвора усиливается в процессе скейлинга.

Изменение рабочей функции затвора оказывает эффективное воздействие на электро-физические характеристики кремниевых полевых GAA нанотранзисторов. Поскольку составляющие исследуемого механизма имеют случайный характер, то адекватная оценка степени их влияния на электро-физические характеристики кремниевых полевых GAA нанотранзисторов с цилиндрической геометрией играет важную роль. Однако, в отличии от других транзисторных архитектур, нет возможности контролировать с помощью дополнительной регулировки влияние неравномерность работы выхода металлического затвора.

Публикация выполнена в рамках НИР ФГУ ФНЦ НИИСИ РАН по теме № FNEF-2024-0003 "Методы разработки аппаратно-программных платформ на основе защищенных и устойчивых к сбоям систем на кристалле и сопроцессоров искусственного интеллекта и обработки сигналов".

Sensitivity of the Potential Distribution in the Channel of Silicon GAA Nanotransistors to the Anomalous Behavior of the Metal Gate Grain

N. Masalsky

Abstract. The influence of the anomalous grain size of the metal gate of a silicon field-effect GAA nanotransistor with cylindrical geometry on the potential distribution in its working area is discussed. Based on the analytical solution of the 2D Poisson equation, an analytical model has been developed to analyze the sensitivity of the potential distribution of silicon field GAA nanotransistors to the abnormal behavior of the metal gate grain. The variations of the potential distribution in transistors with short and thin working areas with a length from 25 to 11 nm are quantitatively analyzed. The dependence of the potential perturbation on the location of the abnormal grain on the gate is shown. The linear dependence of the amplitude of the disturbance on the magnitude of the output operation jump is established. A mathematical model of the fluctuation of the potential distribution has been developed, including variations in the unevenness of the boundaries of the anomalous grain area. The nonuniformity of the boundaries of the anomalous region makes an additional contribution to the transformation of the potential. With approximately the same deformations of the boundaries, the contribution of this mechanism is not significant. With a significant asymmetry of the boundaries, the contribution can exceed 10% of the perturbation generated by the ideal ring.

Keywords: silicon cylindrical nanotransistor with a surrounding gate, anomalous grain size of a metal gate, potential distribution, Poisson's equation, modeling

Литература

1. International Roadmap for Devices and Systems (IRDS), More Moore. 2017. Available online: https://irds.ieee.org/roadmap-2017 (accessed on 15 September 2022).

2. J.S. Yoon, T. Rim, J. Kim, K. Kim, C.K. Baek, Y.H. Jeong. Statistical variability study of random dopant fluctuation on gate-all-around inversion-mode silicon nanowire field-effect transistors // “Appl. Phys. Lett.”, (2015), V. 106, 103507

3. D. Nagy, G. Indalecio, A.J. Garcia-Loureiro, M.A. Elmessary, K. Kalna, N. Seoane. FinFET versus gate-all-around nanowire FET: performance, scaling, and variability. // “IEEE Journal of the Electron Devices Society”, (2018), V. 6, 332-40.

4. M. Onobajo, J. Silva-Martinez. Analog circuit design for process variation-resilient systems-on-a-chip. Dordrecht: Springer, 2012.

5. M.M. Tehranipoor, U. Guin, D. Forte. Counterfeit integrated circuits: Detection and Avoidance. Springer, 2015.

6. I. Ferain, C.A. Colinge, J. Colinge. Multigate transistors as the future of classical metal–oxide–semiconductor field-effect transistors // “Nature”, (2011), V. 479, 310–316.

7. K. Kuhn, C. Kenyon, A. Kornfeld, M. Liu, A. Maheshwari, W.K. Shih, S. Sivakumar, G. Taylor, P. VanDerVoorn, K. Zawadzki. Managing process variation in Intel’s 45nm CMOS technology // “Intel Technology Journal”, (2008), V. 12, 93-109.

8. J. A. Croon, W. Sansen, H.E. Maes. Matching properties of deep sub-micron MOS transistors, Springer, 2005

9. S.K. Saha. Modeling process variability in scaled CMOS technology. // “IEEE Design Test of Computers”, (2010), V. 27, 8–16.

10. K. K Young. Analysis of conduction in fully depleted SOI MOSFETs // “IEEE Trans. Electron Devices”, (1989), V. 36, No. 3, 504-506.

11. Н.В. Масальский. Влияние зернистости металлического затвора кремниевых конических GAA нанотранзисторв на флуктуации порогового напряжения // “Труды НИИСИ РАН”, (2023), Т. 60, № 6, 387-393

12. Н.В. Масальский. Моделирование ВАХ ультратонких КНИ КМОП нанотранзисторов с полностью охватывающим затвором // “Микроэлектроника”, (2021), Т. 60, № 6, 387-393

Особенности разработки органического корпуса для многокристальных сборок на основе чиплетов

А.А. Подковыров[1], А.В. Андреев[2]

[1]НИЦ «Курчатовский институт» – НИИСИ, Москва, Россия, barfey@cs.niisi.ras.ru;

[2]НИЦ «Курчатовский институт» – НИИСИ, Москва, Россия, alandreev@cs.niisi.ras.ru

Аннотация. В работе рассматривается разработка и методика проектирования органического корпуса с шариковыми выводами типа BGA (Ball Grid Array) с двумя кристаллами, соединенных между собой высокоскоростными интерфейсами. Описаны особенности топологии корпуса на основе чиплетов. Рассмотрены основные технологии и этапы разработки. Проведен обзор разработок зарубежных аналогов.

Ключевые слова: чиплет, система в корпусе, корпусирование, технология «перевернутого кристалла», многокристальный модуль

1.Введение

Технология чиплетов в наши дни завоевывает огромную популярность в полупроводниковой промышленности. Чиплеты представляют собой специально разработанные кристаллы, которые способны работать в системе вместе с себе подобными внутри корпуса. Несколько чиплетов представляют собой одну сложную микросхему с общей подложкой. Их использование сокращает время разработки нового устройства и снижает затраты за счет интеграции разработанных чипов в многокристальный модуль (МКМ). Они коммутируют между собой соединениями внутри подложки, а после интерфейсные сигналы проходят к массиву шариковых выводов (BGA). Самыми первыми компаниями-гигантами, которые разработали устройства на базе чиплетов, стали Intel и AMD [1].

Корпус на основе чиплетов разрабатывался на предприятии ФГУ ФНЦ НИИСИ РАН. Он предназначен для отработки технологии создания специализированных СБИС в виде системы в корпусе (СвК), состоящих из чиплетов на подложке и оптимизированных по эффективности и технологической доступности для отечественных дизайн-центров.

2. Система в корпусе

Технология система в корпусе (СвК) представляет собой расположение в одном корпусе нескольких СБИС, в котором они подключены между собой.

СвК имеет ряд преимуществ, помимо интеграции нескольких функций, такие как:

- миниатюризация;
- повышенная надежность, увеличение производительности;
- улучшенная электромагнитная совместимость и целостность сигнала;
- невысокие финансовые и временные затраты на разработку [2].

У этой технологии огромный потенциал, поскольку каждый блок СвК может создаваться отдельной группой специалистов, что сокращает время разработки и позволяет подавать на финальную сборку уже испытанные комплектующие.

Систему обычно подразделяют по пространственным расположениям функциональных блоков внутри корпуса, по которым определяется возможность применения в СвК различных типов конструктивного исполнения компонентов и технологий сборки и монтажа. Классифицируют СвК на следующие группы:

- сборка и монтаж блоков в одной плоскости (2D);
- сборка и монтаж блоков через промежуточную подложку/сигнальный мост (2.5D);
- сборка и монтаж блоков в трехмерном пространстве (3D) [3].

3. Обзор зарубежных аналогов

В массовых продуктах чиплеты появились в 2017 году. Они были использованы в процессорах AMD из линейки Ryzen Threadripper. Два кристалла Zeppelin устанавливались на одной подложке, что способствовало двухкратному увеличению ядер, по сравнению с процессорами Ryzen 1000 серии – с 8 до 16 ядер.

На сегодняшний день количество чиплетов у передовых компаний может достигать двузначных значений. К примеру, AMD изготовило новое поколение серверных процессоров EPYC с 12-ти процессорными чиплетами в корпусе. Вместе с чипом ввода-вывода кристаллов у таких процессоров будет 13, а количество ядер составит 96. Этот процессор построен на архитектуре Zen4 под кодовым названием Genoa (рисунок 1).

Без чиплетной технологии на данный момент столько ядер в одном процессоре реализовать было бы практически невозможно.

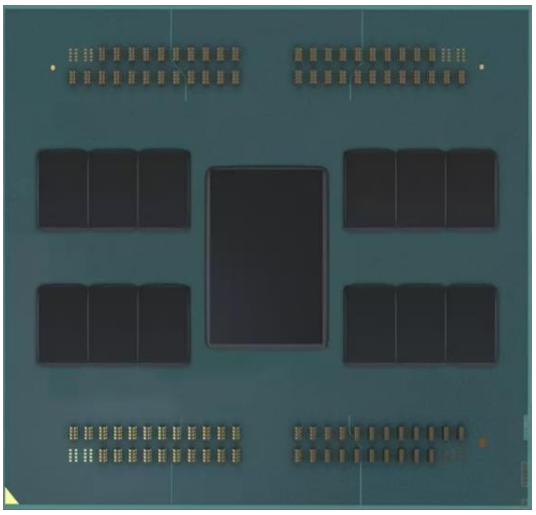

Рис.1. Серверный процессор AMD EPYC на базе 12-чиплетного дизайна

Сейчас AMD применяет одни и те же чиплеты в большинстве линеек своих процессоров. То есть AMD на производстве TSMC просто тиражирует кристаллы, к примеру, по 7-нм нормам для различных изделий, что продемонстрированно на рисунке 2. Чиплетная технология позволяет свободно комбинировать гетерогенные функциональные блоки внутри составной микросхемы в зависимости от её предназначения.

Изготавливать кристаллы настолько небольшого размера крайне выгодно. Повышается выход годных: чем меньше размер, тем выше выход годных. На одной пластине размещается существенно большее число кристаллов, а значит, при равной стоимости изготовления пластины каждый отдельный процессор становится, соответственно, дешевле, о чем еще будет подробнее сказано в главе 5 [4].

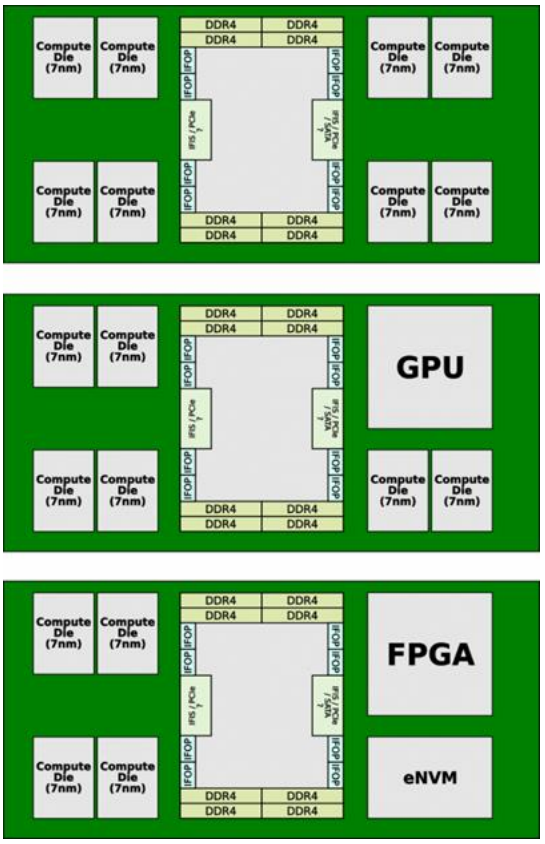

Рис. 2. Различные комбинации чиплетных блоков от AMD

В 2023 году AMD представила первый мобильный чип, оснащённый дополнительной кеш-памятью 3D V-Cache. Процессор имеет 16 ядер с поддержкой 32-х потоков и работает с тактовой частотой до 5,4 ГГц. Базовая частота чипа составляет 2,3 ГГц. Ryzen 9 7945HX3D состоит из трёх чиплетов, два из которых – 8-ядерные на микроархитектуре Zen 4 (Core Complex Die, CCD) и один чиплет ввода-вывода. Технологический процесс 5-нм. На один из CCD установлен кристалл с дополнительной кэш-памятью 3-го уровня (L3) объёмом 64 Мб. С учётом того, что в каждом CCD уже присутствует по 32 Мб кэш-памяти L3, общий её объём составляет 128 Мб, а вместе с кэш-памятью 2-го уровня (L2) суммарный объем доходит до 144 Мб [6,7].

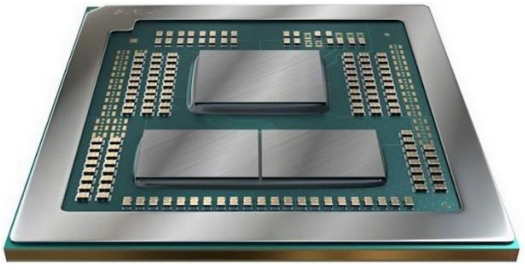

Рис.3. AMD Ryzen 9 7945HX3D

В 2023 году на проходившей в Стэнфордском университете конференции по микропроцессорам «Hot Chips», Intel показала превью своих серверных процессоров Xeon Scalable 6-го поколения Sapphire Rapids (рисунок 4) для демонстрации передовых технологий упаковки. Процессоры с многочиповой архитектурой, предназначенные для центров обработки данных, должны появиться в этом году. Компания в настоящее время проводит тестирование среди своих клиентов.

Процессор Intel Xeon Granite Rapids 6-го поколения имеет структуру с пятью чиплетами: три больших по центру коммутируют между собой через каналы EMIB - встроенный многокристальный межкомпонентный мост. Эти три чипа содержат ядра производительности с 2 Мб кэш-памяти L2, 4 Мб кэш-памяти L3 и четырьмя интерфейсами DDR5. Два меньших чиплета по краям с высокоскоростными интерфейсами ввода-вывода (HSIO). Предполагается, что процессоры Granite Rapids будут иметь 12 каналов памяти DDR5 с поддержкой модулей DDR5-6400 и MCR DIMM, 136 линий PCIe Gen5 с поддержкой CXL 2.0 и до шести каналов UPI.

Ожидается, что серийные процессоры будут иметь от 84 до 90 ядер. Частота работы ядер составляет от 1,1 до 2,70 ГГц в инженерных образцах процессоров.

Чиплеты процессора выпускаются по техпроцессу Intel 3 (3-нм), в то время как чиплеты HSIO производятся на техпроцессе 7-нм для оптимальной производительности и эффективности затрат.

Платформа Intel Granite Rapids может поддерживать от одного до восьми сокетов на сервере [8].

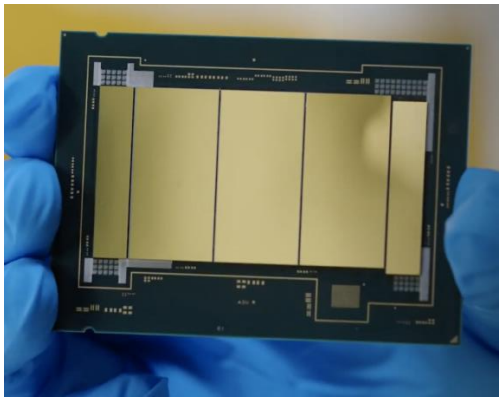

Рис.4. Intel Xeon Granite Rapids 6-го поколения

Чиплеты можно интегрировать как в 2D структуре, как в предыдущих примерах, так и в 3D структуре по методу монтажа «Package-on-Package» (PoP).

Компанией Intel в 2019 году был представлен процессор под кодовым названием Lakefield. Он представляет собой гетерогенное многочиповое решение, объединяющее в модуле одно 10-нм большое высокопроизводительное вычислительное ядро (микроархитекутра Sunny Cove) и четыре 22-нм мелких энергоэффективных ядра (Tremont).

На рисунке 5 указаны основные блоки 10-нм чипа.

Габариты 12 x 12 x 1 мм.

Рис.5. Основные блоки 10-нм чипа Lakefield

Lakefield имеет новую технологию Foveros 3D, которая предусматривает многослойное размещение элементов в одном пакете, что создаёт настоящую систему на чипе (рисунок 6). Каждый слой включает в себя отдельные компоненты, добавленные с помощью 3D-интеграции (рисунок 7).

В процессоре также интегрировано графическое ядро Intel Gen11, обеспечивающее хорошую графическую производительность

Lakefield разрабатывался для использования в мобильных устройствах, таких как ноутбуки, планшеты и гибридные устройства, где энергоэффективность и производительность являются ключевыми характеристиками.

Цель Intel при разработке процессора Lakefield состояла в создании решения, которое бы предлагало эффективное использование вычислительных ресурсов, высокую производительность и долгую автономную работу для мобильных устройств, и это стало возможным благодаря использованию чиплетов и технологии Foveros [5,9,10].

Рис.6. Стековая архитектура Lakefield

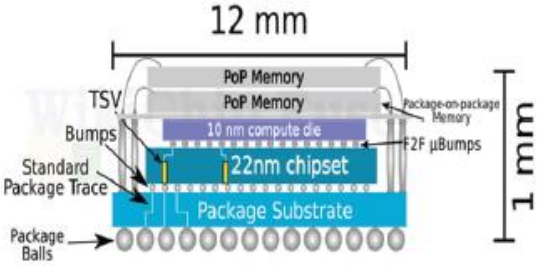

Рис.7. Структура процессора Lakefield

AMD и Intel являются главными конкурентами на мировом рынке процессоров и задают всем странам тренд для развития технологий и усовершенствования в этой области. Гетерогенная интеграционная в корпусе с использованием технологии чиплетов предоставляют альтернативу системе на кристалле, особенно при использования малых технологических норм, которые большинство компаний не могут себе позволить.

4. Технология «перевернутого кристалла» («Flip-chip»)

Flip chip (перевёрнутый чип) — это метод монтажа чипа на печатные платы или другие подложки, при котором кристалл микросхемы устанавливается на выводы, расположенные на его контактных площадках, равномерно распределенных по поверхности кристалла микросхемы. Главной особенностью и одновременно большим преимуществом данной технологии является то, что кристалл монтируется напрямую, без дополнительных соединительных проводов, активной контактной стороной вниз – к подложке.

Эта технология считается самой распространенной, поскольку, на сегодняшний день, ни одна высокоинтегрированная «система в корпусе», либо многокристальная сборка не обходится без применения данного метода установки кристалла.

Этот метод был создан компанией IBM в 1961 и тогда назывался SLT (Solid Logic Technology). Тогда предполагалось использование покрытых припоем медных столбиковых выводов. В 1965 году IBM создала технологию Flip chip C4 (Control Collapse Chip Connection). В Flip chip C4 уже использовались шарики, полностью из припоя. Они подразумевают собой припойные бампы кристалла диаметром от 100 мкм и более. У этого метода есть ряд преимуществ по сравнению с проволочным монтажом wire bonding, такие как:

- минимальные массогабаритные показатели и быстродействие;
- улучшенные электрические характеристики электронных компонентов за счёт коротких соединений проводников;
- меньшее количество соединений, что сокращает количество потенциальных узлов отказа и обеспечивает более эффективное распределение тепловой энергии;
- низкая стоимость материалов;
- малые размеры корпуса;
- большее количество выводов;
- высокая плотность сигнала;
- низкая сигнальная индуктивность и хорошее подключение питания/земли;
- идеально подходит для высокоскоростных интерфейсов;
- лучшее рассеивание мощности.

К недостаткам этой технологии можно отнести высокие значения механических напряжений в контактах, вызванные малой пластичностью выводов. Для их уменьшения, а также повышения прочности и теплоотвода, в 1987 году компания Hitachi впервые применила при flip-chip монтаже сверхтекучий компаунд для инкапсуляции (заполнения пространства между смонтированным кристаллом и подложкой), который называется underfill. На рисунке 8 схематически показаны крепления кристалла при помощи двух

типов монтажа.

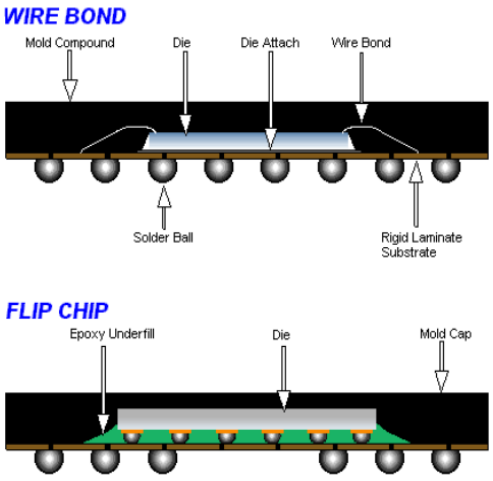

Рис.8. Крепление кристалла в корпусе: с помощью проволочных соединений (wire bonding); методом «перевёрнутого кристалла» («flip-chip»)

В наши дни многие компании используют технологию flip-chip для изготовления миниатюрных и беспроводных электронных устройств. Ее применение позволяет уменьшать размеры микросхемы, например, с чиплетными сборками, а, следовательно, это приводит к увеличению ее применения в таких значимых областях, как искусственный интеллект, военная и аэрокосмическая промышленность, телекоммуникации.

5. Этапы разработки корпуса

Подразделяются на следующие:

- написание ТЗ: на каждый кристалл выдается свое частное техническое задание для корпусирования, связанное с его функциональным назначением;
- на этом этапе подбирается вид корпуса, в зависимости от количества развариваемых выводов и типа монтажа. Если тип монтажа проволочный (wire bond), то подбирается материал проволоки и форма профиля. Так же подбирается материал корпуса в зависимости от назначения будущей микросхемы;
- преемственность топологии при модернизации корпуса: для того, чтобы учесть замену чиплета на другой, нужно предусмотреть использование универсального посадочного места, чтобы без дополнительных затрат и технических решений произвести замену.
- разработка топологии корпуса в соответствии с требованиями используемых интерфейсов и габаритами;
- моделирование - после разработки топологии необходимо промоделировать на целостность сигналов и питания для того, чтобы исключить все неточности и создать надежную микросхему для разных условий эксплуатации;
- реализация - после успешного моделирования на этапе реализации создается КД, где вносятся все необходимые требования по корпусированию в соответствии с требованиями завода-изготовителя;
- тестирование - при получении готовой микросхемы необходимо проверить и протестировать ее в разных условиях эксплуатации, в которых она должна функционировать;
- установка - транспортировка и установка микросхемы на печатную плату;
- функциональное тестирование в составе модуля или целостного устройства.

Главной отличительной особенностью разработки чиплетного корпуса от обычного является создание межсоединений между кристаллами в соответствии с ТЗ на микросхему. Это относится к четвертому пункту – разработке топологии корпуса.

6. Преимущества и недостатки применения чиплетов

Монолитные кристаллы для процессоров достаточно крупные, и при использовании кремниевых пластин типоразмера 300 мм остается достаточно много «лишних» частей, которые потом утилизируются. Таким образом, при использовании меньших кристаллов чиплетов можно задействовать большую площадь пластины, что будет более выгодно в техническом и экономическом плане. Наглядно это можно увидеть на рисунке 9.

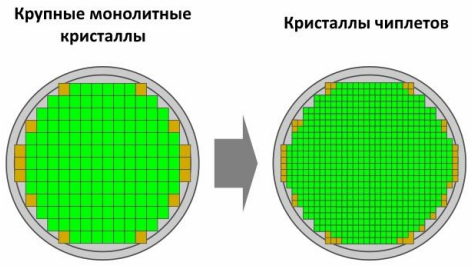

Рис.9. Варианты использования кристаллов

Технология чиплетов дает возможность разрабатывать заменяемые и повторно используемые полупроводниковые блоки, а также в зависимости от того, какие функции целевой микросхемы выполняет чиплет, имеется возможность комбинирования различных техпроцессов в системе из чиплетов. Это значительно снижает затраты на разработку и упрощает тестирование.

При использовании новой технологии поначалу всегда будут свои трудности и недостатки. Например, коммутация чиплетов происходит через проводники внутри подложки, что приводит к дополнительным задержкам при передаче информации.

На рисунке 10 можно хорошо заметить, как

влияет и насколько выгоден переход к чиплетному дизайну, по сравнению с монолитной СБИС [5]. По вертикали указано количество годных микросхем на пластине-заготовке (%). По горизонтали – их площадь (мм2). С ростом габаритов чипа значительно уменьшается выход годных чиплетов.

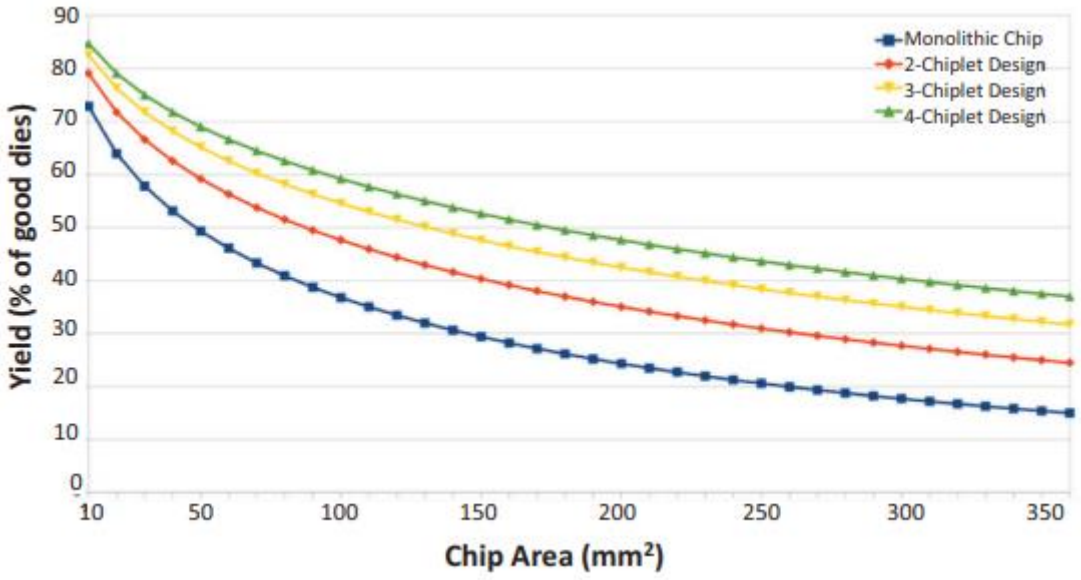

Рис. 10. График отношения площади кристалла к выходу годных

7. Особенности и требования к топологии разрабатываемого корпуса

Одной из особенностей разводки корпуса является применение стеков переходных отверстий через 4 слоя. В разработке были использованы два таких стека: TOP-INT4 для линий интерконнекта и INT1-CORE1 для подключений дифференциальных пар интерфейса TRS, как показано на рисунке 11.

При проектировании топологии корпуса на основе чиплетов в первую очередь в слое TOP и INT1 преимущественно выводятся цепи питания и земли. Для удобства и более равномерной разводки дифференциальных пар, а также для исключения их удлинения, было решено создать симметричную структуру переходных (рисунок 12), что позволяет при зеркальной перестановке двух коротких проводников в слое INT3 быстро изменить полярность подключения на шариковые выводы, как потребуется.

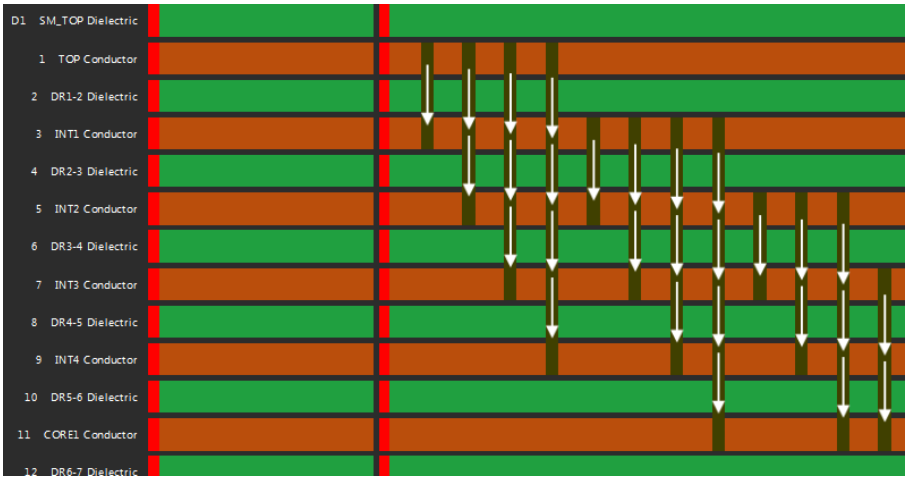

Рис. 11. Структура слоев корпуса для чиплетов

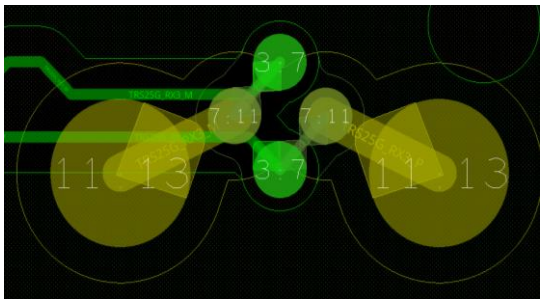

Рис. 12. Симметричная структура переходных для подключения дифференциальных пар

На корпусе микросхемы установлены резисторы с минимальным допуском на калибровочных дифференциальных линиях высокоскоростных портов, которые позволяют:

- сократить длину проводников к резисторам;
- сэкономить полезную площадь в корпусе и печатной плате;
- уменьшить количество шариковых выводов в корпусе и компонентов на плате.

В корпусе установлены два кристалла. На скриншоте (рисунок 13) слева находится ответный чиплет, а центре второй - интерфейсный. Они объединены между собой межсоединительной высокопроизводительной шиной AXI. Сама микросхема содержит следующие интерфейсы, которые выводятся на печатную плату:

- высокоскоростные: PCIe, DDR, HDMI, DPRT;
- интерфейсы для отладки: UART, JTAG, I2C.

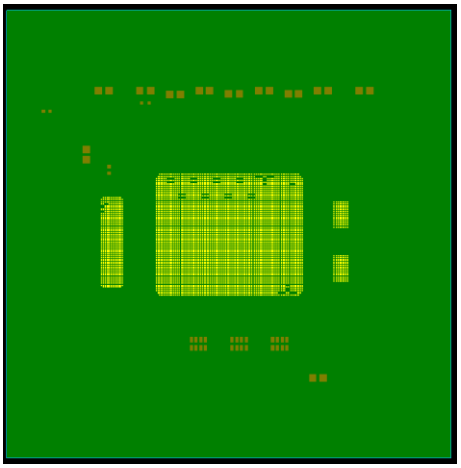

Рис. 13. Скриншот слоя TOP

Количество слоев и их толщина указаны в таблице 1.

Таблица 1. Используемые слои в корпусе

	Название	**Слой**	**Толщина (мкм)**
1	SM_TOP	Mask	10
2	TOP	Conductor	15
3	DR1-2	Dielectric	30
4	INT1	Conductor	15
5	DR2-3	Dielectric	30
6	INT2	Conductor	15
7	DR3-4	Dielectric	30
8	INT3	Conductor	15
9	DR4-5	Dielectric	30
10	INT4	Conductor	15
11	DR5-6	Dielectric	30
12	CORE1	Conductor	15
13	DR6-7	Dielectric	800
14	CORE2	Conductor	15
15	DR7-8	Dielectric	30
16	INT11	Conductor	15
17	DR8-9	Dielectric	30
18	INT12	Conductor	15
19	DR9-10	Dielectric	30
20	INT13	Conductor	15
21	DR10-11	Dielectric	30
22	INT14	Conductor	15
23	DR11-12	Dielectric	30
24	BOTTOM	Conductor	15
25	SM_BOT	Mask	10

8. Заключение

В статье были представлены некоторые особенности топологии органического корпуса для многокристальных сборок на основе чиплетов. Представлены основные технологии, используемые при разработке. Применение корпусов на основе чиплетов позволяет улучшить компактность, увеличить скорость и производительность систем, отвечая вызовам современных вычислительных модулей. Дальнейшее исследование в данной области позволит совершенствовать технологии чиплетов и расширять возможности интеграции элементов в высокотехнологичные системы.

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН по теме FNEF-2024-0003.

Features of the Development of an Organic Package for Multi-Chip Modules Based on Chiplets

A.A. Podkovyrov, A.V. Andreev

Abstract. The paper discusses the development and design methodology of an organic package with BGA (Ball Grid Array) ball leads with two crystals connected by high-speed interfaces. The features of the chiplet topology are described. The main technologies and stages of development are considered. A review of the developments of foreign analogues was carried out.

Keywords: Chiplet, System-in-Package (SiP), packaging, Flip-Chip technology, multi-chip module

Литература

1. Баранов А.М., Подковыров А.А., Андреев А.В. Проблемы и направления развития микросхем с перевёрнутым кристаллом. Труды НИИСИ РАН. 2024. Т. 14. № 1. С. 4-10.

2. Вертянов Д., Сидоренко В., Тимошенков С., Ковалев А. Перспективные конструктивно-технологические решения для производства «систем-в-корпусе» // Технологии в электронной промышленности. 2019. № 4. С. 60–64.

3. В. Мейлицев Системы в корпусе. Краткий обзор технологий // Электроника НТБ. 2021. Выпуск 2. С. 108-113.

4. Бобков С.Г. Технология чиплетов – перспективное направление развития российской микроэлектроники // Электронная техника. Серия 3: Микроэлектроника. 2022. № 1 (185). С. 42-51.

5. Lau, J. H., «State-of-the-Art and Outlooks of Chiplets Heterogeneous Integration and Hybrid Bonding», Journal of Microelectronics and Electronic Packaging (2021) 18, 145–160.

6. AMD представила Ryzen 9 7945HX3D — первый мобильный процессор с дополнительной кеш-памятью 3D V-Cache. URL: https://3dnews.ru/1090674/amd-predstavila-ryzen-9-7945hx3d-perviy-mobilniy-protsessor-s-dopolnitelnoy-keshpamyatyu-3d-vcache (дата обращения 5.06.2024).

7. AMD представила аналог Ryzen 9 7950X3D для ноутбуков — 7945HX3D предлагает 144 МБ кэш-памяти L2+L3. URL: https://overclockers.ru/blog/molexandr/show/100816/amd-predstavila-analog-ryzen-9-7950x3d-dlya-noutbukov-7945hx3d-predlagaet-144-mb-kesh-pamyati-l2-l3 (дата обращения 5.06.2024).

8. Intel представила процессоры Granite Rapids с пятью чиплетами. URL: https://overclockers.ru/blog/andr_83/show/108238/intel-predstavila-processory-granite-rapids-s-pyatju-chipletami (дата обращения 6.06.2024).

9. Intel рассказала о первом гибридном процессоре Lakefield. URL: https://nvworld.ru/news/intel-announced-lakefield/ (дата обращения 6.06.2024).

10. 5-ядерный процессор Intel Lakefield замечен в базе данных 3DMark. URL: https://ru.gecid.com/news/5-yadernyj_processor_intel_lakefield_zamechen_v_3dmark (дата обращения 7.06.2024).

Оценка энергопотребления в программе комплексного тестирования производительности вычислительного кластера

О. И. Вдовикин[1], П. Н. Телегин[2], Б. М. Шабанов[3]

[1]МСЦ РАН – филиал ФГУ ФНЦ НИИСИ РАН, НИЦ «Курчатовский институт», Москва, Россия, vdovikin@jscc.ru;

[2]МСЦ РАН – филиал ФГУ ФНЦ НИИСИ РАН, НИЦ «Курчатовский институт», Москва, Россия, telegin@jscc.ru;

[3]МСЦ РАН – филиал ФГУ ФНЦ НИИСИ РАН, НИЦ «Курчатовский институт», Москва, Россия, shabanov@jscc.ru.

Аннотация. Для сравнительной оценки производительности вычислительных кластеров используются различные группы тестов. Для определения параметров эффективности кластерных решений с большим числом ядер разработана программа комплексного тестирования производительности Laptest. Из-за высокого потребления электроэнергии суперкомпьютерами энергоэффективность при выполнении задач стала являться одним из важнейших свойств вычислительных систем. В статье описывается реализация оценки потребления энергии в программе Laptest.

Ключевые слова: суперкомпьютер, вычислительный кластер, тестирование производительности, энергоэффективность, RAPL

1. Введение

Эффективность работы вычислительных систем зависит от архитектуры систем и базового программного обеспечения (ПО): системного, промежуточного, прикладного. Для каждой архитектуры и ПО производительность определяется классами задач и реализацией прикладных программ. В связи с этим для оценки эффективности систем используются различные группы тестов.

Традиционно наиболее важным показателем производительности вычислительных систем являются операции над 64-разрядными числами. Обычно используются 3 вида тестов: стандартные тесты, программы пользователей и синтетические тесты. При этом тестируются три уровня, представленные в Таблице 1.

Тесты SPEC (Standard Performance Evaluation Corporation [1]) оценивают производительность вычислительной системы. Производительность на операциях с плавающей запятой оценивается наборами тестов SPECfp2000 (14 тестовых программ), SPECfp2006 (17 программ), SPEC CPU2017 (10 тестов, поддерживающих OpenMP).

Таблица 1. Уровни тестов

Уровень	Реализация	Тесты
Операции над структурами данных	Ядро процессора	STREAM Memory Test Suite SPEC
Нити (threads)	Узел кластера	SPEC DGEMM
Процессы	Кластер	NAS HPL программы пользователей

Тесты NAS Parallel Benchmarks [2] оценивают производительность параллельных вычислительных систем на программах из аэрокосмической области, в том числе с использованием среды MPI расширения языков OpenMP.

Тест HPL [3] используется при составлении рейтингов TOP500 [4], и TOP50 [5]. Тест основан на решении системы линейных уравнений методом LU-разложения. HPL использует среду MPI, библиотеку линейной алгебры BLAS [6] и применяется для сравнения суперкомпьютеров [7]. В рейтинг TOP500 также используется тест HPCG (High Performance Conjugate Gradients) [8]

для работы с разреженными матрицами, что характерно для многих реальных задач [9]. Тем не менее в списке Top500 представлены результаты только для 118 суперкомпьютеров из 500.

Важное значение для производительности имеет то, как система работает с оперативной памятью. Тесты STREAM, Memory Test Suite выполняют как простые операции чтения/записи, так и операции с векторной обработкой данных [10].

Подпрограмма умножения матриц DGEMM из библиотеки BLAS также часто используется в качестве теста, так как эффективно использует возможности аппаратуры и реализована практически для всех платформ.

В набор HPC Challenge Benchmark [11] были собраны тесты, наиболее часто используемые для оценки и анализа компонентов высокопроизводительных систем, включая вычислительные тесты (HPL, DGEMM, FFT), тесты работы с памятью и межпроцессорных коммуникаций.

Стоит также отметить, что в связи с бурным развитием технологий искусственного интеллекта был разработан тест HPL-MxP (Mixed-Precision Benchmark) [12], использующий в работе 16-разрядные числа. При этом результаты использования теста существенно зависят от входных данных [13].

Из-за большой электрической мощности суперкомпьютеров все более актуальными являются вопросы энергопотребления [14]. Например, в списке Top500 отдельно ведется список Green500 – наиболее энергоэффективных суперкомпьютеров. В набор тестов SPEC CPU2017 добавлено измерение максимальной и средней мощности, а также потребленной задачами энергии.

Главным недостатком стандартных тестов является то, что задачи пользователей, как правило, не укладываются в рамки тестов ни по используемым алгоритмам, ни по реализации.

2. Программа комплексной оценки производительности вычислительного кластера

В МСЦ РАН создавались вычислительные системы серии МВС: МВС-1000М, МВС-15000ВМ, МВС-6000IM, МВС-100К, МВС-10П/*. При выборе системных решений одним из важнейших параметров были результаты тестов производительности вычислительных платформ. В разработанной методике выбора суперкомпьютерных платформ предусмотрено тестирование компонентов до создания вычислительных систем [15].

Стандартные тесты не всегда дают необходимые данные для выбора платформ, так как реальные программы вариативны. Поэтому был разработан тест LAPTEST [16], оценивающий стандартные действия на компонентах вычислительный системы, реализующий методику [15]. Наиболее близкий тест – это HPC Challenge Benchmark. Однако, потребовалась большая полнота тестирования [17]. Кроме того, для имитации реальной нагрузки желательно иметь возможность обеспечивать одновременно разную загрузку вычислительных узлов и их групп. Тест ориентирован на анализ вычислительных кластеров с многоядерными процессорами.

Стандартные действия разделены на типовые элементы по тестированию как оперативной памяти, так и связки процессор-память, дисковых операций, сетевых обменов, загрузки процессора.

Рассмотрим детальнее.

Тестирование производительности памяти. Возможны следующие варианты выполнения:

- чтение;
- запись;
- одновременные чтение/запись.

Операции могут выполняться над регулярными или иррегулярными данными, с одним или двумя потоками данных при чтении.

Тестирование производительности коммуникационной сети с использованием среды MPI. Варианты выполнения:

- передача данных (send);
- прием данных (recv);
- одновременные передача/прием (sendrecv);
- широковещательная передача данных (bcast).

Тестирование производительности при работе с дисковой подсистемой. Варианты выполнения:

- чтение;
- запись;
- одновременные чтение/запись.

Тестирование производительности ядер под нагрузкой. Вызывается подпрограмма DGEMM. Данная подпрограмма сильно оптимизирована практически для всех вычислительных платформ, поэтому показывает близкую к максимально возможной производительность, а также задействует значительную часть исполнительных устройств процессора, что повышает его температуру и позволяет, кроме прочего, оценить реализацию систем охлаждения. Возможно тестирование с использованием внешних подключаемых библиотек. В настоящее время реализован интерфейс для одновременного использования до восьми реализаций DGEMM. Библиотечные функции настраиваются в конфигурационном файле.

Ожидание (sleep). Используется для организации простоя процессора.

Тест предназначен для оценки характеристик вычислительных кластеров с большим числом ядер, в том числе коэффициентов замедлений при одновременной работе различного числа ядер.

Тесты имеют гибкую структуру и настраиваются с помощью конфигурационного файла. В нем можно задавать набор тестов, их раскладку по узлам и ядрам, варианты тестирования, длительность и объем данных.

3. Измерение энергопотребления

Так как рост производительности вычислительных систем носит не только интенсивный, но и экстенсивный характер, становится необходимым кроме измерения параметров, непосредственно описывающих производительность, оценивать также и энергоэффективность вычислительных систем. В связи с этим в синтетический тест Laptest были добавлены средства измерения энергопотребления при выполнении как задач, так и отдельных операций.

3.1. Способы измерения энергопотребления

В настоящее время кроме «аналогового» способа измерения энергопотребления с использованием ваттметра или токовых клещей, производителями высокопроизводительных систем стали использоваться интегрированные в платформу варианты для измерения потребления как системы в целом, так и отдельных ее компонент, таких как центральный микропроцессор и оперативная память. Прикладное ПО, работающее в среде ОС Linux, имеет возможность получить данные об энергопотреблении несколькими способами [18], например:

- с использованием интерфейсов Running Average Power Limit (RAPL);
- прямым чтением специальных регистров микропроцессоров (MSR);
- с использованием библиотек Performance Application Programming Interface (PAPI).
- посредством обращения к Baseboard Management Controller (BMC) или Intelligent Platform Management Interface (IPMI).

Наиболее простым и универсальным вариантом является использование программных интерфейсов RAPL, который реализуется одной из подсистем ОС Linux. Он обеспечивает платформонезависимый доступ к показателям энергопотребления и предоставляет информацию о потребленной энергии, измеряемой в мДж. При этом измерение производится косвенно при помощи программной модели и основано на данных специальных регистров (MSR).

RAPL также используется и в PAPI [19] как один методов для получения данных о энергопотреблении.

В свою очередь IPMI и BMC являются стандартизованными механизмами для мониторинга системы, которые кроме энергопотребления измеряют также температуру, обороты вращения вентиляторов, напряжение и статус других компонент платформы.

3.2. Реализация оценки потребления энергии

Технически ОС Linux предоставляет доступ к RAPL через специальные файлы, расположенные в файловой системе /sys. На используемых в МСЦ РАН системах можно получить данные для четырех счетчиков энергопотребления: двух микропроцессоров и двух банков памяти. Для каждого из них используются специальные файлы energy_uj. Поскольку счетчики работают в накопительном режиме и имеют фиксированную разрядность, система предоставляет возможность получения максимального значения до переполнения (max_energy_range_uj), когда счетчик сбрасывается в ноль.

В прототипе Laptest добавлен новый модуль, который обеспечивает измерение потребляемой энергии во время выполнения тестов. В конфигурационном файле помимо параметров запуска тестов указывается список опрашиваемых счетчиков. В начале и конце работы каждого из тестов всеми выделенными для работы процессами осуществляется опрос датчиков энергопотребления и передача их «нулевому» процессу, где данные суммируются для всех датчиков, а затем вычисляется среднее значение потребленной мощности, которое попадает в отчет о работе теста. Для упрощения обработки данных и исключения необходимости модификации самих тестов, продолжительность их работы выбрана таким образом, чтобы исключить двойное переполнение счетчиков энергопотребления [20].

3.3. Оценка корректности измерений

Для оценки корректности измерения энергопотребления во время работы тестов одновременно произведено получение данных энергопотребления через IPMI для оперативной памяти и микропроцессоров:

ipmitool -c -b 0x06 -t 0x2c nm statistics power domain memory:

ipmitool -c -b 0x06 -t 0x2c nm statistics power domain cpu.

С помощью shell-скрипта данные IPMI получались каждые 2 секунды, а затем усреднялись для каждого из тестов.

3.4. Результаты измерений

Тестовые прогоны проводились на узлах, использующих два микропроцессора Intel Xeon E5-2697A (Broadwell) и 128 ГБ оперативной памяти, а также два микропроцессора Intel Xeon Gold 6154 (Skylake) и 192 ГБ оперативной памяти.

Были выбраны следующие тесты Laptest:

- S – Sleep – процессы «спят» заданное в конфигурации время;
- MR1 и MR2 – Memory Read – все ядра (MR1) или потоки (MR2) осуществляют одновременное чтение последовательных данных из оперативной памяти;
- MW1 и MW2 – Memory Write – все ядра (MW1) или потоки (MW2) осуществляют одновременную последовательную запись данных в оперативную память;
- BURN1 и BURN2 – Burn - все ядра (BURN1) или потоки (BURN2) одновременно выполняют операцию DGEMM над матрицами.

Полученные в результате тестовых прогонов данные представлены ниже (Таблицы 2 и 3).

Таблица 2. Результат запуска на узле Broadwell

Тест	Средняя потребляемая мощность (RAPL), Вт	Средняя потребляемая мощность (IPMI), Вт
S	39	38
MR1	259	258
MR2	282	281
MW1	250	249
MW2	265	263
BURN1	277	275
BURN2	326	322

Таблица 3. Результат запуска на узле Skylake

Тест	Средняя потребляемая мощность (RAPL), Вт	Средняя потребляемая мощность (IPMI), Вт
S	68	73
MR1	349	351
MR2	386	386
MW1	370	370
MW2	420	421
BURN1	371	373
BURN2	437	436

Таким образом, тестовые прогоны показали приемлемую точность измерений, проводимых с помощью данных, получаемых через интерфейсы RAPL, в сравнении с аналогичными, полученными через IPMI.

Публикация выполнена в рамках государственного задания по проведению фундаментальных исследований по теме FNEF-2024-0016.

Заключение

В программу комплексного тестирования производительности вычислительного кластера добавлена возможность оценки потребляемой мощности вычислительных узлов. Результаты исследования показали корректность и приемлемую точность измерений, проводимых с помощью реализованных средств.

Estimation of Power Consumption in a Comprehensive Computing Cluster Performance Testing Program

Oleg Vdovikin, Pavel Telegin, Boris Shabanov

Abstract. Different groups of tests are used to compare the performance of computing clusters. The Laptest comprehensive performance testing program was developed to determine the efficiency parameters of the computing cluster systems with a large number of cores. Due to the high energy consumption of supercomputers, energy efficiency of executing tasks has become one of the most important features of computing systems. The article describes the implementation of energy consumption estimation in the Laptest program.

Keywords: supercomputer, computing cluster, performance testing, energy efficiency, RAPL

Литература

1. Standard Performance Evaluation Corporation, 2024. URL: http://www.spec.org/. (Дата обращения: 18 07 2024).

2. NAS Parallel Benchmarks. URL: http://www.nas.nasa.gov/Resources/Software/npb.html. (Дата обращения: 18 06 2024).

3. A. Petitet, R. C. Whaley, J. Dongarra и A. Cleary. HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers, 2016. URL: http://www.netlib.org/benchmark/hpl/. (Дата обращения: 04 08 2018).

4. TOP500. The List, 1993-2024. URL: https://www.top500.org/. (Дата обращения: 17 08 2024).

5. Суперкомпьютеры top50, 2023. URL: http://top50.supercomputers.ru/?page=rating. (Дата обращения: 15 08 2024).

6. BLAS (Basic Linear Algebra Subprograms) 2024. URL: https://www.openblas.net/. (Дата обращения: 23 07 2024).

7. D. Papakyriakou, I. S. B. Barbounakis. High Performance Linpack (HPL) Benchmark on Raspberry Pi 4B (8GB) Beowulf Cluster. «International Journal of Computer Applications», Т. 185 (2023) №. 25, 11-19.

8. J. Dongarra, M. A. Heroux, P. Luszczek. HPCG Benchmark: a New Metric for Ranking High Performance Computing Systems. Technical Report, Electrical Engineering and Computer Sciente Department, UT-EECS-15-736, Knoxville, Tennessee, 2015.

9. А.В. Киселев, Е. А. Киселев и В. В. Корнеев. Анализ структуры информационного графа теста HPCG. «Научный сервис в сети Интернет: многообразие суперкомпьютерных миров. Труды Международной суперкомпьютерной конференции (22-27 сентября 2014 г., г. Новороссийск)» М.: Изд-во МГУ, 2014, 49-51.

10. Memory Test Suite 2020. URL: https://openbenchmarking.org/suite/pts/memory. (Дата обращения: 17 07 2024).

11. The HPC Challenge (HPCC) benchmark suite. ACM, Inc., 2024. URL: https://dl.acm.org/doi/10.1145/1188455.1188677. (Дата обращения: 18 07 2024).

12. rocHPL-MxP. 2024. URL: https://github.com/ROCm/rocHPL-MxP. (Дата обращения: 19 07 2024).

13. G. Henry, E. Petit, A. Lyashevsky, P. Caday. Deconstructing HPL-MxP Benchmark: A Numerical Perspective. «Euro-Par 2024: Parallel Processing». Berlin, Heidelberg, Springer-Verlag, 2024, pp. 47 - 60.

14. S. Devineni и G. Bhargavi. Energy-Efficient Computing and Green Computing Techniques. «Computer Science, Engineering and Technology» T.1 (2023), № 4, 37-45.

15. Б.М. Шабанов, Исследование, разработка и применение суперкомпьютерных вычислительных систем. Докторская диссертация. Москва, 2019, http://www.frccsc.ru/sites/default/files/docs/ds/002-073-02/diss/08-shabanov/ds02-08-shabanov_main.pdf?336. (Дата обращения: 25 12 2019).

16. М.С. Клинов, С. Ю. Лапшина, П. Н. Телегин, Б. М. Шабанов. Особенности использования многоядерных процессоров в научных вычислениях. «Вестник УГАТУ». Т. 1 (2012), № 6(51), 25-31.

17. Б.М. Шабанов, П. Н. Телегин, О. С. Аладышев. Особенности использования многоядерных процессоров. «Программные продукты и системы». (2008) № 2, 7-9, 2008.

18. F. Alizadeh Moghaddam, T. Geenen, P. Lago, P. Grosso. A user perspective on energy profiling tools in large scale computing environments «2015 Sustainable Internet and ICT for Sustainability (SustainIT)». Madrid, Spain, 2015, 1-5.

19. V.M. Weaver, M. Johnson, K. Kasichayanu, J. Ralph, P. Luszczek, D. Terpstra, S. Moore. Measuring Energy and Power with PAPI. «2012 41st International Conference on Parallel Processing Workshops». Pittsburgh, PA, USA 2012, 262-268.

20. K.N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, Z. Ou. RAPL in Action: Experiences in Using RAPL for Power Measurements. «ACM Transactions on Modeling and Performance Evaluation of Computing Systems». T.3 (2018), №2, Article 9, 1-26.

Оценка влияния различных участков программного кода на энергопотребление вычислительной системы

Е. А. Киселёв[1], Д. А. Чубаров[2], А. В. Баранов[3]

[1] МСЦ РАН – филиал ФГУ ФНЦ НИИСИ РАН, НИЦ «Курчатовский институт», Москва, Россия, kiselev@jscc.ru, +7(926)223-88-66;

[2] РТУ МИРЭА, Москва, Россия, chubarovdima@inbox.ru;

[3] МСЦ РАН – филиал ФГУ ФНЦ НИИСИ РАН, НИЦ «Курчатовский институт», Москва, Россия, anton.baranov@jscc.ru, +7(903)215-43-42.

Аннотация. Работа посвящена проверке гипотезы о возможности оценки влияния исходного кода программы на энергопотребление вычислительной системы. На основе исследованных методов оптимизации авторами предложен алгоритм для оценки энергоэффективности программного кода. Разработан макет программного средства в виде расширения для Visual Studio Code, реализующий представленный алгоритм. Приведены экспериментальные результаты по исследованию различных способов повышения энергоэффективности программного кода, а также результаты проверки работоспособности разработанного алгоритма.

Ключевые слова: энергоэффективность, анализ исходного кода, оптимизация программного кода, RAPL, VS Code

1. Введение

С каждым годом расширяется область применения высокопроизводительных вычислительных систем, возрастает их энергопотребление. С увеличением числа процессорных ядер в таких системах растет степень параллелизма, появляются его новые уровни, такие как векторная обработка, вычисления на графических процессорах или иных ускорителях. Усложняются параллельные алгоритмы, и пользователи суперкомпьютеров вместо разработки собственных алгоритмов и программ предпочитают применять готовые программные пакеты и коды. В этой связи перестают работать подходы к оптимизации параллельных программ, основанные на частичной или полной переработке параллельного алгоритма. Ряд проблем, например, наличие неустранённых зависимостей при векторизации программного кода, не могут быть решены без нарушения алгоритма работы программы [1]. При этом следует отметить, что для таких задач применение различных техник оптимизации программного кода позволяет не только значительно сократить время выполнения параллельного приложения, но и добиться сокращения энергопотребления вычислительной системы, на котором оно было запущено.

В настоящей работе мы сфокусировали внимание на исследовании влияния различных стратегий оптимизации программного кода на энергопотребление вычислительной системы. Нами предложен подход к выявлению участков программного кода, оказывающих наибольшее влияние на энергопотребление вычислительной системы. Отличительной особенностью предлагаемого подхода является отсутствие необходимости запуска программы на выполнение и, как следствие, сокращение затрат на лишние тестовые запуски.

Наше исследование можно условно разделить на две части. Первая часть посвящена анализу подходов к оптимизации программного обеспечения и экспериментальной оценке их влияния на энергопотребление. Полученные результаты позволяют оценить не только влияние различных стратегий оптимизации на энергоэффективность, но и энергозатраты на выполнение часто используемых вычислительных операций. Во второй части исследования представлен алгоритм автоматического выявления наиболее энергоемких участков программного кода, требующих наибольшего внимания со стороны разработчика. Для демонстрации работы предложенного алгоритма было разработано расширение для среды разработки Visual Studio Code и продемонстрирована его работа на примере двух тестовых параллельных программ.

2. Способы и алгоритмы повышения энергоэффективности программного кода

Для сокращения энергопотребления вычислительных систем при выполнении программ применяются два основных подхода. Первый предполагает оптимизацию программного кода (Energy-Aware Programming) [2,3], второй – оптимизацию выделения вычислительных ресурсов для выполнения каждого приложения (Energy Aware Scheduling) [4]. Преимуществом первого подхода является возможность заранее (до запуска программы) выявить участки программного кода, которые будут оказывать значительное влияние на энергопотребление, и оптимизировать их. Основным недостатком этого подхода является необходимость в наличии исходного кода программы. Преимуществом второго подхода является возможность сокращения энергопотребления на основе накопленных данных статистики использования ресурсов вычислительной системы без модификации исходного кода. Основным недостатком здесь является необходимость выполнения серии запусков программы для сбора статистических данных. При этом энергопотребление программы на одних и тех же вычислительных ресурсах может отличаться в зависимости от параметров запуска (например, числа используемых потоков).

Для оптимизации программного кода используют следующие различные стратегии.

1. Рефакторинг программного кода;
2. Оптимизация алгоритма программы;
3. Оптимизация на этапе компиляции программы.

Рефакторинг кода – это процесс переработки исходного кода программы путем изменения его внутренней структуры без изменения внешнего поведения с целью улучшения удобочитаемости, надежности, безопасности и производительности. В последнее время именно рефакторинг чаще всего применяется для повышения энергоэффективности программ. Большой вклад в исследование и развитие этого направления внесли работы [2, 3]. Авторами отмечено, что применение различных методов рефакторинга позволяет значительно сократить энергопотребление мобильных устройств под управлением операционной системы Android. В работе [4] продемонстрировано, как использование различных методов рефакторинга влияет на энергопотребление Java-программ. Результаты аналогичных исследований, но для большего числа методов рефакторинга и для программ на языке C++ представлены в работе [5]. Общим для всех работ является вывод о том, что рефакторинг программного кода может приводить как к сокращению, так и увеличению энергопотребления. При этом определено, что вне зависимости от приложений, могут быть выделены энергоэффективные методы рефакторинга [5]. В работе [6] исследовано влияние комбинаций методов рефакторинга кода Java и C#-программ для мобильных и настольных систем. Установлено, что не для всех случаев комбинирование энергоэффективных методов рефакторинга приводит к сокращению энергопотребления. Авторы отмечают необходимость продолжения исследования различных комбинаций методов рефакторинга, выявления совместимых друг с другом, а также продолжения работ по совершенствованию программных средств измерения энергопотребления.

В работе [7] авторы продемонстрировали влияние эффективности алгоритма программы на энергопотребление вычислительной системы. В ходе эксперимента была произведена сортировка числовых элементов пузырьковым алгоритмом с вычислительной сложностью $O(n^2)$ и алгоритмом «пирамидальной (кучной) сортировки» вычислительной сложности $O(n \times \log(n))$. В результате алгоритм «пирамидальной (кучной) сортировки» потребил в 1,5 раза меньше энергии, чем алгоритм «пузырьковой сортировки». В работе [8] авторы для решения головоломки «Ханойские башни» использовали рекурсивный и итеративный алгоритмы. Итеративный вариант решения головоломки приводил к увеличению энергопотребления в 5,14 раза по сравнению с рекурсивным. Результаты таких экспериментов демонстрируют существенное влияние алгоритма на энергопотребление вычислительной системы.

Проектирование циклов с соблюдением некоторых простых правил позволяет существенно снизить влияние их выполнения на энергопотребление вычислительной системы. Например, использование беззнаковых целочисленных счетчиков или нуля в качестве условия завершения обратного отсчета ускоряет выполнения цикла за счет использования меньшего количества регистров микропроцессора. Энергопотребление может быть снижено путем разворачивания циклов – объединения инструкций, которые вызываются в нескольких итерациях цикла, в одну итерацию. Разворачивание циклов снижает точность предиктора ветвлений, поскольку существует меньше ветвей, на которых предиктор может обучить свое поведение. Однако это также снижает частоту прерывания непрерывного потока последовательных выборок, так что в качестве совокупного эффекта потребление энергии на инструкцию уменьшается.

Циклы «вращения» и «опроса» являются другими потенциальными источниками повышенного энергопотребления. В циклах «вращения» процесс или поток многократно проверяет, является ли условие истинным, например, доступен ли ввод с клавиатуры или блокировка. Когда поток находится в цикле «вращения», он остается активным, не выполняя полезную задачу. Использование циклов «вращения» может быть эффективным, если потоки заблокированы на короткие периоды времени. Таким образом можно избежать накладных расходов от перепланирования процесса операционной системы или переключения контекста. В качестве альтернативы применяется метод «опроса». В этом случае поток переходит в спящее состояние до тех пор, пока не произойдет какое-либо событие. Программисту необходимо найти компромисс между экономией энергии, получаемой за счет более длительного пребывания в состоянии с пониженным энергопотреблением, и накладными расходами, возникающими из-за перепланирования или частых переходов состояний.

Операционные системы предоставляют программистам механизм управления потоками через интерфейс системных вызовов. Использование нескольких потоков и нескольких ядер обеспечивает лучшую производительность, ускоряет вычислительный процесс и, как следствие, сокращает совокупные затраты на энергопотребление [9].

Векторизация программного кода и использование расширенного набора инструкций, таких как SIMD, также позволяют ускорить вычисления и сократить энергопотребление.

Оптимизация на этапе компиляции программного кода осуществляется в автоматическом режиме с использованием оптимизирующих компиляторов, которые выполняют анализ и преобразования программ на разных уровнях абстракции, начиная от исходного кода, промежуточного кода, такого как трехадресный код, до ассемблерного и машинного кода. Анализы и преобразования могут иметь разные области действия. Они могут выполняться в пределах одного базового блока (локальные), между базовыми блоками, внутри процедуры (глобальные) или между границами процедур (межпроцедурные). Традиционно оптимизирующие компиляторы пытаются сократить общее время выполнения программы или использование ресурсов, таких как память. Сам процесс компиляции может быть выполнен до выполнения программы (статическая компиляция) или во время выполнения программы (динамическая компиляция). Это большое пространство проектирования является основной проблемой для разработчиков компиляторов. Необходимо рассмотреть множество компромиссных вариантов для того, чтобы оправдать разработку и реализацию конкретного прохода или стратегии оптимизации.

3. Результаты исследования способов увеличения энергоэффективности программного кода

С целью проверки влияния оптимизации программного кода на его энергоэффективность в рамках настоящего исследования проведена экспериментальная оценка наиболее результативных методов оптимизации программного кода.

Первая серия экспериментов была направлена на проверку гипотезы из работы [10] о том, что для копирования массива эффективнее использовать встроенные функции вместо циклов. Вторая серия экспериментов была посвящена проверке другой гипотезы из работы [10] о том, что локальность при обращении к данным может снизить общее энергопотребление системы.
Третья серия экспериментов производилась с целью проверки гипотезы R4 [11], в которой предложено использовать единственный цикл вместо нескольких вложенных для сокращения энергопотребления.

Так как в [11] рассматривались не только отдельные методы оптимизации программного кода, но и совокупность этих методов, было принято решение в четвертой серии экспериментов оценить совокупность способов под обозначением C4, которая включает в себя: замену ссылки на саму переменную, добавление переменных в код для упрощения конструкций и удаление вложенных циклов.

Приведённые в таблице 1 результаты экспериментов демонстрируют эффективность перечисленных выше методов. Отдельно стоит выделить эффективность способа, связанного с векторизацией.

4. Влияние типовых операций на энергопотребление

Рассмотрим оценку влияния на энергопотребление часто используемых типовых вычислительных операций. В работе [10] было показано, что центральный процессор (ЦП) является бóльшим потребителем электроэнергии по сравнению с оперативной памятью (ОП) и дисковой подсистемой. Однако в [10] не показано, какие операции, выполняемые ЦП, являются наиболее энергозатратными. Также следует отметить, что в указанной работе эксперименты проводились с твердотельным накопителем (SDD), а не с жест-

ким диском (HDD), обладающим меньшей скоростью и большим энергопотреблением.

С целью определения влияния различных операций на энергопотребление ЦП нами была проведена серия измерений. В ходе каждого измерения исследуемые операции (сложение, разность, деление, умножение) повторялись в цикле $10^6, 10^7, 10^9, 10^{11}, 10^{13}$ раз. Полученные значения были усреднены и представлены на графиках (рис. 1-4).

В ходе проведения экспериментов было замечено, что с определённого момента потребление энергии перестаёт резко расти. Это связано с имеющимися ограничениями ЦП и невозможностью одновременного выполнения требуемого количества вычислительных операций. При большом количестве однотипных операций образуется очередь на обработку, что приводит к замедлению роста энергопотребления и увеличению времени обработки всех операций. По этой причине рост энергопотребления замедляется.

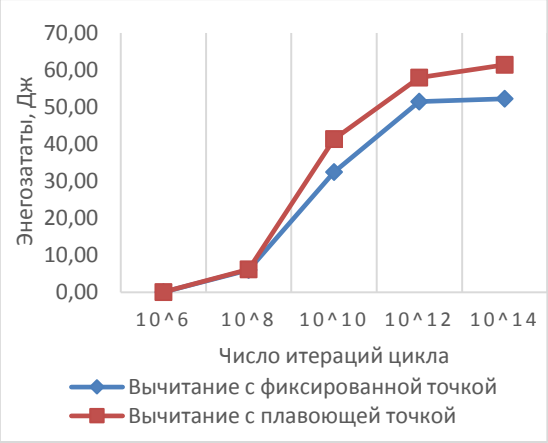

Рис. 1. Энергозатраты на выполнение операций разности с плавающей и фиксированной точкой

Рассмотрим результаты эксперимента по оценке энергозатрат при выполнении операций чтения файла, приведенные на рисунках 5 и 6. На графиках видно, что при обращении к жёсткому диску (рис. 5) расходуется значительно больше энергии по сравнению с арифметическими операциями. В этом случае на энергопотребление оказывает влияние не только частота обращения к жесткому диску, но и объём данных (рис. 6). Энергозатраты на обращение к оперативной памяти сопоставимы с арифметическими операциями (рис. 7).

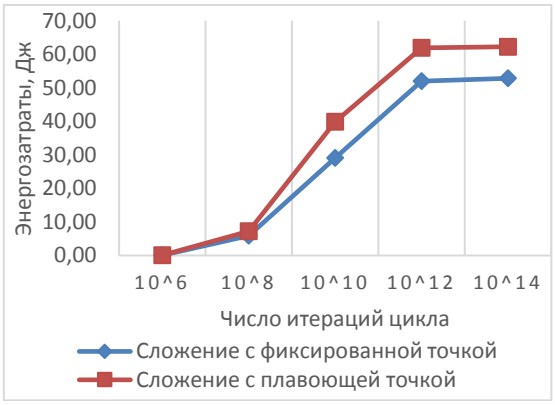

Рис. 2. Энергозатраты на выполнение операций сложения с плавающей и фиксированной точкой

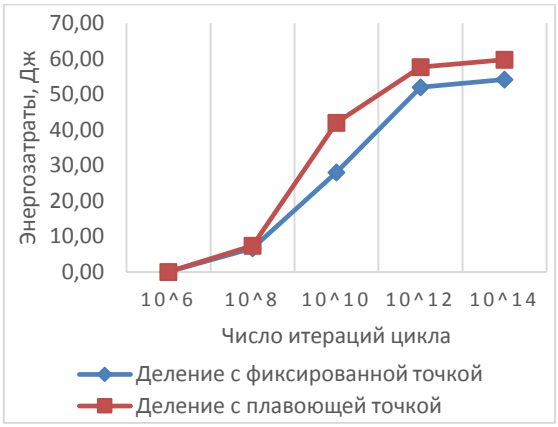

Рис.3. Энергозатраты на выполнение операций деления с плавающей и фиксированной точкой

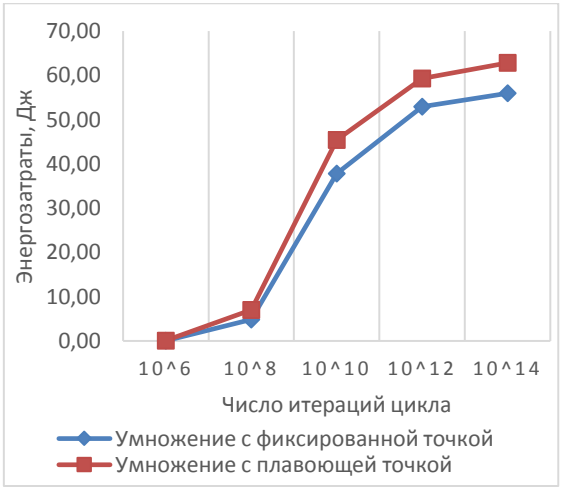

Рис.4. Энергозатраты на выполнение операций умножения с плавающей и фиксированной точкой

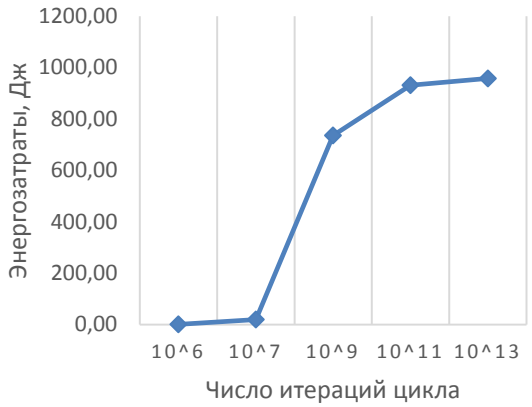

Рис. 5. Энергозатраты на операцию записи символа в файл в зависимости от числа итераций

Рис. 6. Энергозатраты на операцию чтения данных из файла

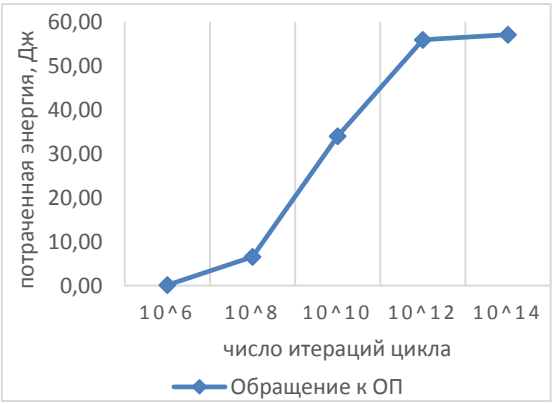

Рис. 7. Энергозатраты на запись символа в оперативную память в зависимости от числа итераций

5. Алгоритм оценки энергоэффективности исходного кода программ

Анализ результатов экспериментов позволяет сделать вывод о практической возможности выявления энергозатратных операций до запуска программы. Например, участки программного кода с вложенными циклами for или while, операции обращения к файловой системе, операции над числами с плавающей точкой и другие операции могут быть отранжированы в соответствии с их энергозатратностью. Полученные данные могут быть использованы при компиляции на этапе семантического анализа программного кода или в среде разработки на этапе профилирования и отладки программы.

С учётом полученных данных нами предложен алгоритм выявления участков программного кода с повышенным энергопотреблением, блок-схема которого представлена на рисунке 8.

Рассмотрим основные шаги алгоритма.

Шаг 1. Инициализировать переданные в качестве входных аргументов множества:

- множество O включает в себя упорядоченный набор конструкций (операции с фиксированной, плавающей точкой, операции доступа к ОП, HDD, циклы и др.), которые будут использоваться для анализа;

- множество W включает в себя коэффициенты, которые соответствуют конструкциям из множества O (например, операциям с фиксированной, плавающей точкой и операции доступа к ОП может соответствовать 1, а операциям доступа к HDD и циклам, как более энергозатраным – 2 и т.д.);

- множество S включает в себя набор строк программного кода, которые будут анализироваться.

Шаг 2. Выполнить N итераций цикла, где N – число элементов во множестве S. На каждой итерации цикла выполнить шаги 2.1-2.3.

Шаг 2.1. Взять очередную строку из множества S и найти соответствующую ей конструкцию из множества O. Отметить, если строка является началом или концом цикла.

Шаг 2.2. Найденной на шаге 2.1 конструкции определить число из множества W, присвоить его текущей строке из множества S и перейти к шагу 2.3.

Шаг 2.3. Проверить, находится ли текущая строка из множества S в цикле (в том числе и во вложенных циклах). Если строка входит в цикл, то к присвоенному на шаге 2.2 значению прибавляется значение всех циклов, в которые она входит.

Таким образом, после окончания работы алгоритма на выходе формируется модифицированное множество S, где каждой строке соответствует число, характеризующее ее энергозатраты.

Алгоритм является сходящимся, так как число его итераций зависит от числа объектов в конечном множестве S. Поскольку в алгоритме реализован однократный обход всех строк кода, его вычислительная сложность не превышает O(n), где n – количество строк кода.

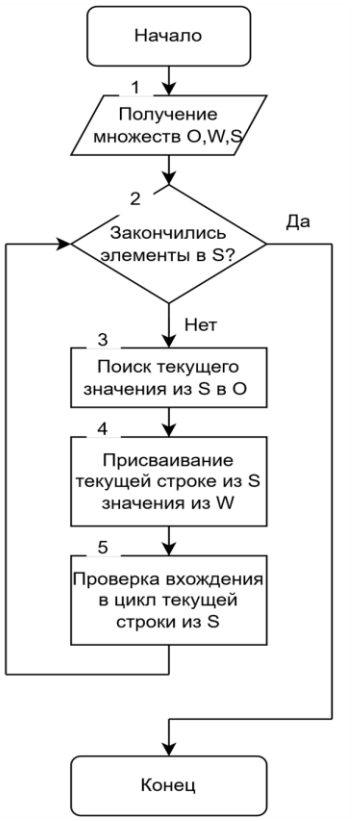

Рис. 8. Блок-схема алгоритма определения влияния заданного участка кода на энергопотребление

6. Проверка алгоритма

Предложенный алгоритм был реализован в виде расширения редактора программного кода Visual Studio Code (далее – VS Code) [12]. Разработанное расширение, названное Energy Mapper, включает в себя модуль анализа программного кода и модуль визуализации результатов анализа. Схема взаимодействия программных модулей приведена на рисунке 9.

Для работы с разработанными модулями пользователь должен открыть интересующий его файл в среде разработки VS Code и запустить Energy Mapper.

Открытый файл передаётся на вход модуля анализа программного кода, который возвращает коэффициенты для каждой строки в соответствие с шагами 2.2 и 2.3 алгоритма. Модуль визуализации получает эти коэффициенты и в соответствии с ними подсвечивает строки программного кода в среде разработки.

Проверка работоспособности модулей была проведена при помощи программ, реализующих алгоритмы быстрой сортировки и решения задачи Коши. Сначала было проведено измерение энергопотребления при запуске программ без изменений программного кода, а затем – с оптимизацией энергозатратных участков кода, отмеченных в Energy Mapper, как показано на рисунке 10. Измерения значений энергопотребления производилось с помощью библиотеки RAPL Stopwatch [13].

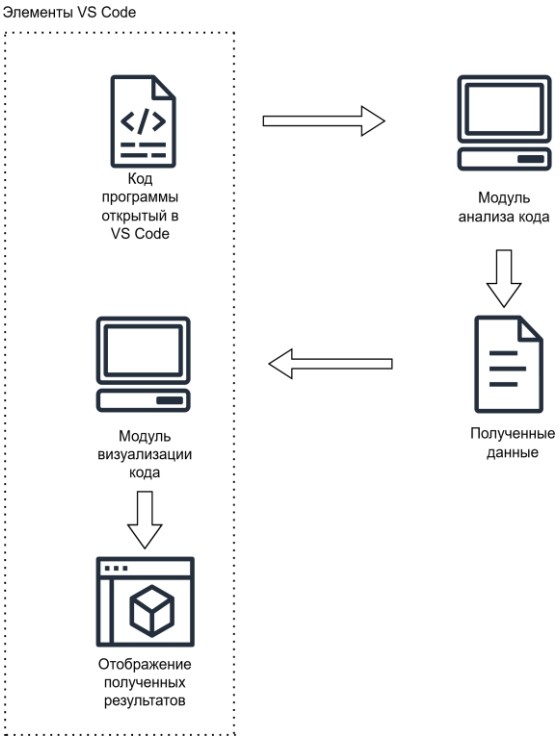

Рис. 9. Структура разработанных модулей

Рис. 10. Демонстрация фрагмента программного кода, размеченного модулем визуализации

Из результатов, приведённых в таблице 2, видно, что удалось достичь уменьшения энерго-

потребления при выполнении тестовых программ. При этом в эксперименте, связанным с задачей быстрой сортировки, удалось достичь более «скромных» результатов по сравнению с задачей Коши. Это связано с тем, что в программном коде алгоритма быстрой сортировки присутствует энергозатратная функция генерации псевдослучайных чисел rand().

Разработанное расширение Energy Mapper предоставляет возможность при помощи конфигурационных настроек задавать собственные новые сущности (функции или методы) для анализа.

Заключение

Результаты проведенных исследований позволяют сделать ряд выводов.

Во-первых, рефакторинг программного кода позволяет снизить накладные расходы на время выполнения параллельной программы и ее влияние на энергопотребление вне зависимости от выбранного языка программирования. Могут быть выделены языковые конструкции и функции оказывающие наибольшее влияния на энергопотребление. По мнению авторов, представляется целесообразным провести их классификацию.

Во-вторых, предложенные в настоящей работе алгоритм и программное средство Energy Mapper демонстрируют практическую возможность оценки энергозатрат на выполнение программ до их запуска на вычислительной системе.

В-третьих, результаты проведенных нами экспериментов с программной реализацией алгоритмов быстрой сортировки и решения задачи Коши продемонстрировали целесообразность использования расширения Energy Mapper для оптимизации исходных текстов программ.

Работа выполнена в рамках государственного задания по теме FNEF-2024-0016.

Таблица 1. Оценка влияния оптимизации кода на энергоэффективность

Способы оптимизации программного кода	**Изменение мощности в %**	**Изменение времени в %**	**Изменение потреблённой энергии в %**
Использование функции копирования данных вместо цикла	+14.09	-86.01	-84.11
Использование локальности при обращении к данным	+2.95	-10.73	-8.12
Использование векторизация вместо последовательного выполнения	+12.61	-98.48	-98.3
Использование совокупности способов оптимизации программного кода	+9.07	-98.66	-98.59

Таблица 2. Оценка влияния оптимизации кода на энергоэффективность

	Потреблённая энергия кода (Дж)	**Изменение потреблённой энергии**
Задача быстрой сортировки	689134	0%
Оптимизированная задача быстрой сортировки	588767	-14.6%
Задача Коши	527951	0%
Оптимизированная задача Коши	173301	-67.1%

Assessing of the Impact of Source Code Different Sections on the Computing System Power Consumption

E. A. Kiselev, D. A. Chubarov, A. V. Baranov

Abstract. The hypothesis about the possibility of assessing the impact of the program source code on the computing system power consumption is tested in the work. The authors proposed an algorithm for assessing the program code energy efficiency based on the studied optimization methods. The software tool prototype as an extension for Visual Studio Code implementing the presented algorithm was developed. Experimental results on the study of various ways to improve the program code energy efficiency, as well as the results of testing the developed algorithm operability are presented.

Keywords: energy efficiency, code analysis, code optimization, RAPL, VS Code

Литература

1. Юрченко А. В. Проектирование и анализ программного обеспечения с низким энергопотреблением с помощью программных метрик энергоэффективности // Машиностроение и компьютерные технологии. 2013. №1. С. 215-234.

2. da Silva, W.G.; Brisolara, L.; Corrêa, U.B.; Carro, L. Evaluation of the impact of code refactoring on embedded software efficiency. In Proceedings of the 1st Workshop de Sistemas Embarcados, Gramado, Brazil, 24–28 May 2010; pp. 145–150.

3. Gottschalk M., Jelschen J., Winter A. Energy-efficient code by refactoring. Softwaretechnik-Trends 2013, 33, 23–24.

4. Sahin, C.; Pollock, L.; Clause, J. How do code refactorings affect energy usage? In Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Torino, Italy, 18–19 September 2014; pp. 1–10.

5. Park, J.J.; Hong, J.E.; Lee, S.H. Investigation for Software Power Consumption of Code Refactoring. In Proceedings of the Twenty Sixth International Conference on Software Engineering and Knowledge Engineering (SEKE), Vancouver, BC, Canada, 1–3 July 2014; pp. 717–722.

6. Hayri Acar, Gülfem I Alptekin, Jean-Patrick Gelas, Parisa Ghodous. The Impact of Source Code in Software on Power Consumption // International Journal of Electronic Business Management, Vol. 14, pp. 42-52 (2016).

7. T. Johann, M. Dick, S. Naumann, and E. Kern, “How to measure energy-efficiency of software: Metrics and measurement results” in Green and Sustainable Software (GREENS), 2012 First International Workshop on, 2012, pp. 51–54.

8. A. Noureddine, A. Bourdon, R. Rouvoy, and L. Seinturier, “A preliminary study of the impact of software engineering on GreenIT,” in Green and Sustainable Software (GREENS), 2012 First International Workshop on, 2012, pp. 21–27.

9. Y. D. Liu, “Energy-efficient synchronization through program patterns,” in Green and Sustainable Software (GREENS), 2012 First International Workshop on, 2012, pp. 35–40.

10. Hayri Acar, Gülfem I Alptekin, Jean-Patrick Gelas, Parisa Ghodous. The Impact of Source Code in Software on Power Consumption // International Journal of Electronic Business Management. 2016. №14. С. 42-52.

11. İbrahim Şanlıalp, Muhammed Maruf Öztürk, Tuncay Yiğit. Energy Efficiency Analysis of Code Refactoring Techniques for Green and Sustainable Software in Portable Devices // Electronics. 2022. №11. С. 442-459.

12. Visual Studio Code Getting Started [электронный ресурс] // Microsoft URL: https://code.visualstudio.com/docs/ (дата обращения 26.10.2024)

13. Github. The RAPL Stopwatch library [электронный ресурс] // https://github.com/LorienLV/rapl_stopwatch?ysclid=m2qc5vo4zz264735644 (дата обращения 26.10.2024).

Симулятор системы управления суперкомпьютерными заданиями с внешним интерфейсом управления

Д. С. Ляховец[1], А. В. Баранов[2], А. Ю. Кудрин[3]

[1]МСЦ РАН – филиал ФГУ ФНЦ НИИСИ РАН, НИЦ «Курчатовский институт», Москва, Россия, anetto@inbox.ru;

[2] МСЦ РАН – филиал ФГУ ФНЦ НИИСИ РАН, НИЦ «Курчатовский институт», Москва, Россия, abaranov@jscc.ru;

[3] МСЦ РАН – филиал ФГУ ФНЦ НИИСИ РАН, Москва, Россия, akudrin2002@gmail.com

Аннотация. Одним из востребованных инструментов исследования поведения систем управления суперкомпьютерными заданиями (СУЗ), как сложных систем коллективного пользования, является имитационное моделирование при помощи симуляторов. В статье сформулированы требования к симулятору высокопроизводительной вычислительной системы, включающей в свой состав территориально распределенные суперкомпьютеры. Соответствие выдвинутым требованиям может быть обеспечено за счет реализации внешнего интерфейса управления симулятором. В статье представлен анализ характеристик современных симуляторов СУЗ с точки зрения выдвинутых требований, предложена архитектура симулятора СУЗ с внешним симулятором управления. Рассмотрены первые результаты применения симулятора Elytra, реализующего предложенную архитектуру.

Ключевые слова: высокопроизводительные вычисления, имитационное моделирование, планирование заданий, система управления заданиями

1. Введение

Современные системы высокопроизводительных вычислений представляют собой сложные технологические комплексы, в состав которых может входить множество территориально распределенных вычислительных установок (суперкомпьютеров). Для проведения расчетов в такой вычислительной системе пользователь формирует так называемое задание – информационный объект, включающий расчетную программу, требования к ресурсам и исходные данные. Расчетная программа реализует прикладной параллельный алгоритм, который может быть выполнен на некотором множестве вычислительных узлов (процессорных ядер) суперкомпьютера. Минимальный набор требований к ресурсам определяется числом необходимых заданию узлов (процессорных ядер) и временем выполнения расчетной программы.

Сформированные пользователями задания образуют входной поток, поступающий в специализированную программную систему управления заданиями (СУЗ). СУЗ ведет одну или несколько очередей заданий и выполняет функцию их планирования, т.е. составления расписания запусков заданий на узлах суперкомпьютера. Современные СУЗ являются сложными многопараметрическими системами, поведение которых существенно зависит от значений конфигурационных параметров и характеристик входных потоков заданий, причем эту зависимость практически невозможно определить аналитически. В случае, когда СУЗ управляет заданиями в территориально распределенной системе (ТРС) высокопроизводительных вычислений, сложность системы только возрастает: в составе СУЗ, например, может появиться несколько уровней иерархии управления [1]. В этой связи для исследования алгоритмов планирования СУЗ и определения их оптимальных параметров широко применяется имитационное моделирование [2].

Имитационное моделирование подразумевает создание модели или симулятора СУЗ, повторяющих характеристики реальной системы. Моделирование позволяет исследовать поведение СУЗ без вмешательства в работу отдельного суперкомпьютера или ТРС в целом.

В соответствии с [1] в ТРС выделяют два уровня иерархии управления заданиями. На верхнем уровне ведется глобальная очередь, задания из которой в соответствии с некоторым алгоритмом планирования распределяются на нижний уровень иерархии, состоящий из очередей т.н. локальных СУЗ. Каждая локальная СУЗ управляет одной вычислительной установкой (суперкомпьютером) из состава ТРС и может принимать как задания из глобальной очереди,

так и непосредственно от пользователей своего суперкомпьютера. В результате образуются несколько входных потоков заданий: один глобальный и множество локальных.

Имитационная модель ТРС в этом случае естественным образом составляется из совокупности симуляторов глобальной и локальных СУЗ. При этом для модели ТРС необходимо обеспечить выполнение ряда принципов:

− реализовать работу с динамическим входным потоком заданий, то есть позволять добавлять и удалять задания из очередей СУЗ во время работы симуляторов;

− обеспечить доступ к очереди локальной СУЗ во время моделирования;

− обеспечить синхронизацию модельного времени между всеми экземплярами симуляторов СУЗ, участвующих в моделировании ТРС.

Указанные принципы реализуются в виде некоторого протокола, который будем называть внешним интерфейсом управления симулятора СУЗ. Внешний интерфейс управления позволит реализовать имитационную модель ТРС на основе симуляторов СУЗ.

2. Сравнительный анализ существующих симуляторов

Можно выделить два класса симуляторов: симуляторы распределённых систем и симуляторы локальных СУЗ. Симуляторы распределённых систем можно адаптировать для моделирования ТРС, но в таких симуляторах на уровне локальной СУЗ реализованы собственные алгоритмы планирования, что может негативно повлиять на точность моделирования.

Перспективным является построение симулятора ТРС на основе симуляторов СУЗ, для которых проведено исследование точности воспроизведения моделируемой СУЗ.

Сфокусируем внимание на двух важных аспекта работы симулятора. Во-первых, проанализируем, соответствуют ли существующие симуляторы выдвинутым требованиям. Во-вторых, рассмотрим, как разработчики того или иного симулятора валидировали свой продукт, то есть сравнивали его с реальной системой или другими симуляторами.

2.1. Симуляторы распределённых систем

Нами проанализированы следующие симуляторы распределённых систем:

– Optorsim [3];
– GSSIM [4];
– WorkflowSim [5];
– GroudSim [6].

Симулятор Optorsim разработан для моделирования грид-систем с упором на задания с интенсивными процессами обмена между вычислительными узлами. Центральной настройкой симулятора является матрица пропускной способности между вычислительными узлами.

В литературе не удалось найти сведений о валидации Optorsim. К существенным ограничениям Optorsim следует отнести работу только со статическим входным потоком, а также отсутствие возможности доступа к локальной очереди заданий во время симуляции и синхронизации модельного времени.

Программная платформа GSSIM создана в 2011 году и в значительной мере похожа на модель, пригодную для моделирования ТРС. В ней возможно реализовать как собственный алгоритм планирования уровня ТРС (реализация глобальной очереди, broker scheduling plug-in в терминологии авторов), так и алгоритм планирования уровня СУЗ (scheduling plug-in).

Как и в случае с Optorsim, сведений о валидации GSSIM нет, а сам симулятор GSSIM поддерживает работу только со статическим входным потоком. Судя по описанию GSSIM, у алгоритма планирования уровня ТРС есть доступ к локальной очереди заданий, реализована синхронизация модельного времени. При этом веб-сайт GSSIM не работает, и мы не смогли найти исходные тексты какой-либо версии кода.

Симулятор WorkflowSim расширяет функционал платформы CloudSim, предназначенной для имитации обработки потока заданий в облачных средах. WorkflowSim предлагается использовать для исследования планирования заданий в распределённых системах с учётом накладных расходов, связанных с отказами компонентов вычислительной системы. Симулятор является моделью ТРС и не основан на валидированном симуляторе СУЗ. При этом алгоритм планирования уровня ТРС необходимо выбрать из списка доступных, и, судя по описанию, нельзя заменить на самостоятельно реализованный. В модели используется событийное моделирование, всё взаимодействие внутри неё происходит через очереди сообщений. Это позволяет организовать единое модельное время для всей системы. В статье [5] нет сведений о возможности доступа к локальной очереди заданий СУЗ. Динамический входной поток заданий не поддерживается. Для валидации симулятора авторы на наборе из 10 422 заданий проанализировали точность, как отношение предсказанного моделью общего времени работы к реальному времени работы из журнала СУЗ. В 2021 году на официальной странице симулятора появилось предупреждение, что симулятор более не поддерживается.

Симулятор GroudSim является инструментом для имитации научных процессов и сосредоточен на моделировании и анализе алгоритмов планирования в грид-системах. Для синхронизации модельного времени авторы использовали механизм дискретных событий, что позволяет интегрировать этот симулятор в модель ТРС. Симулятор не предоставляет возможности работы с локальной очередью извне модели, а также возможность работы с динамическим входным потоком заданий. Для валидации авторы провели эксперименты по сравнению симулятора с GridSim по общему времени моделирования на нескольких наборах данных.

2.2. Симуляторы СУЗ

Нами проанализированы следующие симуляторы СУЗ:

– Batsim [7];

– Performance Prediction Toolkit (PPT) [8];

– Alea [9].

Симулятор Batsim создан для исследования различных планировщиков, и во главу угла авторами поставлена валидация самого симулятора с СУЗ OAR [10]. Авторы оформили алгоритм планирования из OAR как плагин для Batsim и провели серию экспериментов на 9 сгенерированных входных потоках, обработанных на реальной системе и в симуляторе Batsim. Авторы визуально сравнивали диаграмму Гантта расписания запусков заданий, анализировали следующие графики сравнения работы симулятора и реальной системы для каждого задания:

− разница времени выполнения заданий;

− разница времени поступления задания в очередь;

− разница времени полной обработки задания (turnaround time) с момента поступления задания в очередь до завершения выполнения.

Для времени полной обработки задания в дополнение к графику для каждого заданий был построен статистический график с плотностью вероятности значения времени полной обработки.

Batsim работает только со статическим входным потоком. В Batsim не реализованы доступ к локальной очереди заданий во время симуляции и синхронизация модельного времени.

Симулятор PPT является параллельным дискретно-событийным симулятором, написанным на Python и предназначенным для быстрого анализа и прогнозирования производительности научных приложений в суперкомпьютерах. В PPT реализованы несколько алгоритмов планирования, такие как First Come First Served, Shortest Job First и другие. Алгоритм Backfill не реализован и отмечен в направлениях дальнейших исследований. Симулятор PPT работает только со статическим входным потоком заданий. Модельное время и доступ к локальной очереди заданий не реализованы. Для валидации авторы провели серию экспериментов по сравнению с симулятором PYSS (Python Scheduler Simulator), для которого нет опубликованных статей и информации о его валидации. При обработке результатов эксперимента авторы визуально сравнивают графики загрузки вычислительных ресурсов и числа запущенных заданий для алгоритма First Come First Served.

Одним из широко используемых решений для моделирования СУЗ в настоящее время является симулятор Alea. В работе [11] с помощью симулятора Alea исследуется влияние скорректированного заказанного времени (soft walltime) на характеристики системы. Позднее в работе [12] тот же авторский коллектив анализирует результаты внедрения исследованного ранее механизма soft walltime с предсказанным временем на грид-системе MetaCentrum в Чехии. Направлением дальнейших работ в [12] указано более детальное исследование других вариантов предсказания времени на симуляторе. Другие исследователи на базе Alea изучают системы пакетирования заданий [13]. В работе [14] представлено расширение Alea, позволяющее оценивать характеристики предоперационного планирования исследования пациентов с использованием ультразвука для лечения онкологических заболеваний.

На вход симулятора подаётся журнал работы реального суперкомпьютера в специализированном формате Standard Workflow Format с характеристиками заданий. Центральной частью симулятора является планировщик, в котором реализован внешний алгоритм планирования заданий. Алгоритм планирования можно выбрать из набора встроенных (FCFS, Shortest Job First, EASY backfilling, Conservative backfilling и другие) или реализовать самостоятельно, соблюдая предлагаемую спецификацию. Планировщик определяет время запуска каждого задания на виртуальном вычислителе, параметры которого также задаются пользователем. В результате работы Alea формирует выходной файл с построенным расписанием и предоставляет различные графики, такие как построенное расписание, средняя загрузка вычислительных ресурсов, число заданий в очереди и на вычислителе и другие. По нашему опыту, для входных потоков с большим числом заданий (порядка нескольких тысяч) графики Alea перестают корректно строиться.

Разработчики Alea не проводили валидацию своего симулятора, так как предлагают собственные реализации алгоритмов планирования, не связанные с планировщиками СУЗ. В работе

[15] мы проводили сравнение точности моделирования симулятора Alea, симулятора отечественной Системы управления прохождением параллельных заданий (СУППЗ) [16] с виртуальным вычислителем и реального журнала работы суперкомпьютера.

Работа с динамическим входным потоком (dynamic workload) заявлена в статье [9], но в инструкции пользователя самого проекта Alea этот функционал отсутствует, как и примеры его использования в конфигурационном файле. В статье под динамическим входным потоком подразумевается возможность увеличения или уменьшения интервала времени до поступления следующего задания в зависимости от производительности планировщика, что не вполне удовлетворяет описанному нами принципу работы с динамическим входным потоком заданий. В Alea не реализованы доступ к локальной очереди заданий во время симуляции и синхронизация модельного времени.

Результаты сравнительного анализа представлены в таблице 1. На его основе авторами было принято решение создать собственную модель СУЗ с внешним интерфейсом управления.

Таблица 1. Результаты сравнительного анализа симуляторов

Симулятор	Динамический входной поток	Получение состояние очереди	Синхронизация модельного времени	Год обновления (статья/код)
Alea	Заявлено, но не работает	Не реализовано	Не реализовано	2020/2023
Batsim	Не реализовано	Не реализовано	Не реализовано	2017/2024
Optorsim	Не реализовано	Не реализовано	Не реализовано	2016/2015
PPT	Не реализовано	Не реализовано	Не реализовано	2017/2021
GSSIM	Не реализовано	Реализовано	Реализовано	2011/2014
WorkflowSim	Не реализовано	Не реализовано	Реализовано	2012/2015
GroudSim	Не реализовано	Не реализовано	Реализовано	2011/не доступен

3. Симулятор Elytra

Разработанный нами симулятор получил название Elytra. При его создании мы использовали опыт, полученный при использовании симулятора Alea. Например, в работе [17] в симуляторе Alea был реализован алгоритм пакетирования заданий. За время длительной эксплуатации симулятора Alea нами был накоплен набор инструментов для обработки выходного потока заданий Alea, позволяющих рассчитать требуемые характеристики и визуализировать различные графики.

В качестве входного потока Alea использует файл формата SWF [18], в котором для каждого задания указываются следующие основные характеристики:

- идентификатор задания;
- время поступления задания в очередь;
- число процессорных ядер, необходимых для выполнения задания;
- заказанное время выполнения задания;
- реальное время выполнения задания, которое меньше или равно заказанному времени выполнения;
- идентификатор пользователя.

Все эти характеристики доступны алгоритму планирования, но при этом их использование не обязательно. Так, встроенные в Alea алгоритмы планирования игнорируют заказанное время выполнения задания, а ориентируются только на реальное время.

Основным результатом моделирования являются рассчитанное для каждого задания время ожидания в очереди и, соответственно, время запуска задания на вычислителе.

Архитектура симулятора Elytra представлена на рисунке 1. Модуль ожидания заданий входного потока позволяет реализовать динамический входной поток заданий, то есть добавление и удаление заданий из очереди модели СУЗ во время работы симулятора. Для удобства модуль поддерживает два режима работы: со статическим входным потоком заданий, при котором все задания поступают в начальный момент времени, и с динамическим входным потоком, при котором задания поступают в моменты времени, определённые внешним источником. Модуль доступа к очереди заданий реализует возможности

просмотра текущей загрузки симулятора локальной СУЗ и добавления задания в очередь во время моделирования.

Модуль управления модельным временем позволяет синхронизировать модельное время с внешним миром. С помощью библиотеки SimPy на языке программирования Python нами был реализован способ моделирования, основанный на событийной обработке заданий.

В Elytra нами были реализованы два алгоритма: обычная очередь (FCFS) и алгоритм обратного заполнения (backfill).

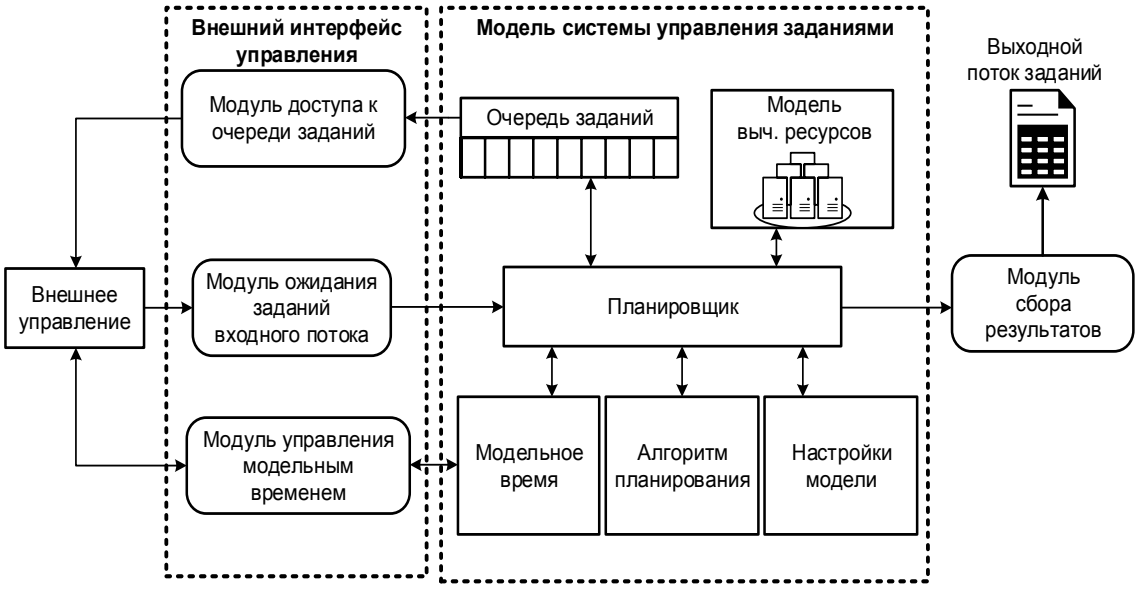

Рис. 1. Архитектура симулятора Elytra

Архитектура симулятора Elytra позволяет реализовать модель TPC с единым модельным временем, с доступом к очереди заданий каждого из симуляторов локальной СУЗ и обменом заданиями между симуляторами за счёт возможности добавления или удаления заданий в процессе моделирования.

4. Экспериментальное исследование симулятора Elytra

Разработанный симулятор после формирования выходного потока заданий рассчитывает ряд характеристик, таких как среднее время ожидания заданий в очереди, среднюю загрузку вычислительных ресурсов и среднюю длину очереди. Для каждой из этих характеристик симулятор формирует график её изменения по времени. Кроме того, симулятор формирует HTML-страницу с планом запуска заданий.

На рисунке 2 представлен пример плана запуска для 10 заданий. По вертикали указаны вычислительные ресурсы, по горизонтали – время от начала эксперимента. Цветные прямоугольники соответствуют заданию, и по клику доступна информация об идентификаторе задания, временах поступления в очередь, старта, завершения, реальном времени выполнения задания и заказанных вычислительных ресурсах. Внутри прямоугольника указывается номер задания. В настройках возможно отключать отображение номеров заданий, что актуально для плана запуска с большим числом заданий.

Для проверки работоспособности и характеристик симулятора Elytra был сгенерирован входной поток из 5000 заданий средней интенсивности на основе подхода, рассмотренного в статье [19]. В этом входном потоке задания поступают в течение 100 часов (4-х с небольшим дней). Моделировалось планирование заданий с помощью реализованных алгоритмов FCFS и Backfill.

На рисунке 3 представлено расписание запусков заданий для исследуемого входного потока заданий. Видно, что в первый день после начала эксперимента в плане запуска множество свободных мест (окон). Это так называемый недогруженный режим работы, во время которого заданий недостаточно для заполнения всех вычислительных ресурсов. В начале 5-го дня задания во входном потоке заканчиваются, после чего в плане запуска снова начинают образовываться окна, связанные с недостатком заданий для заполнения всех ресурсов.

Рис. 2. Пример расписания запусков 10 заданий

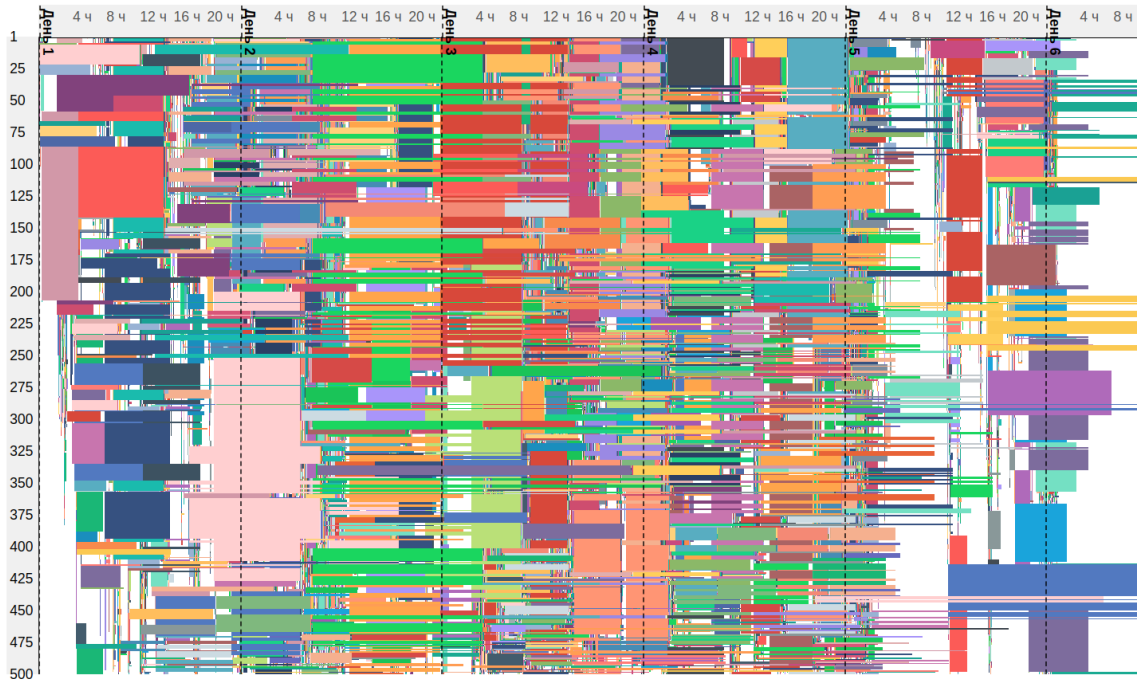

Рис. 3. Расписание запусков заданий для входного потока из 5000 заданий

На рисунке 4 представлен график загрузки вычислительных ресурсов во время эксперимента. В первый день заданий недостаточно для полной загрузки вычислителя. Дни со 2-го по 4-й соответствуют устоявшемуся режиму работы симулятора, когда вычислительные ресурсы интенсивно используются для обработки потока заданий. Видно, что алгоритм backfill обеспечивает бóльшую загрузку вычислителя.

На рисунке 5 представлен график числа заданий в очереди. В 1-й день очереди практически нет, далее очередь некоторое время нарастает. В начале 5-го дня, когда входной поток завершается, очередь быстро пустеет. Видно, что для алгоритма backfill очередь заданий меньше, чем для алгоритма FCFS.

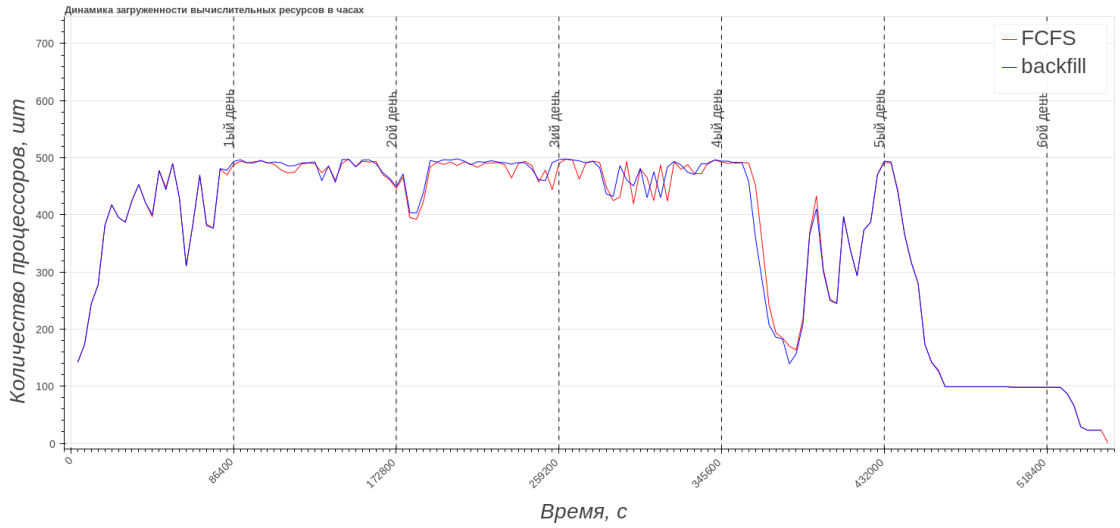

Рис. 4. График загрузки вычислительных ресурсов в Elytra для двух алгоритмов FCFS и backfill

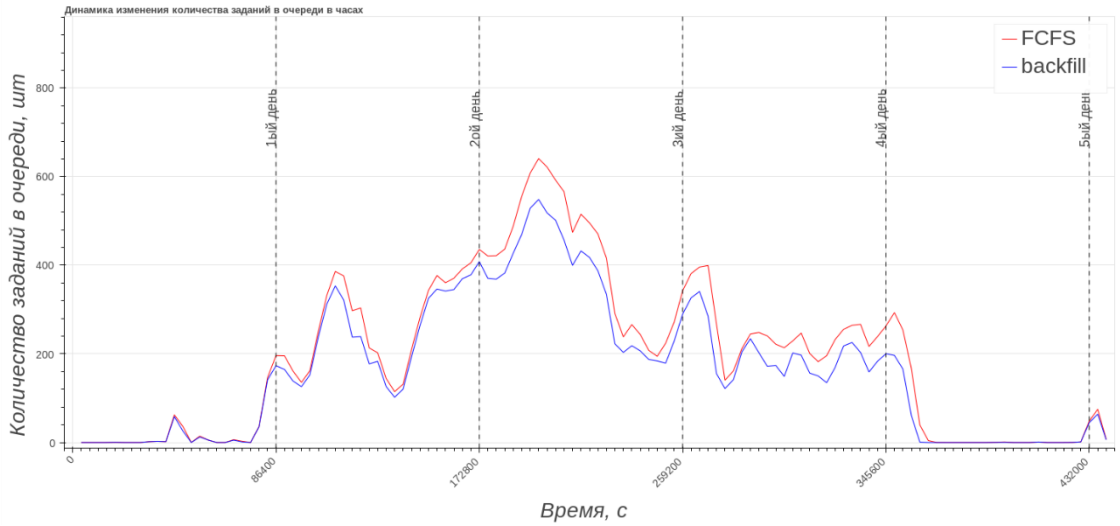

Рис. 5. График числа заданий в очереди Elytra для двух алгоритмов FCFS и backfill

В таблице 1 представлены некоторые из рассчитываемых симулятором характеристик заданий.

Таблица 1. Характеристики выходного потока заданий эксперимента с симулятором Elytra

Характеристика/ Алгоритм	**FCFS**	**Backfill**
Средняя загрузка вычислителя, %	95,2	96,0
Среднее время ожидания задания в очереди, часов	14,8	12,5
Среднее число заданий в очереди	196,1	165,1

Была проверена работоспособность внешнего интерфейса управления. Во время эксперимента симулятор Elytra реагирует на внешнее управление временем, обеспечивает доступ к состоянию очереди заданий, перепланирует задания при добавлении нового задания или удаления из очереди старого, но не запущенного задания. Мы исследовали вопрос снятия с выполнения уже запущенного задания, но такое изменение повлечёт за собой изменение реального времени обработки задания.

Работа симулятора по обработке входного потока проводится быстро, недели вычислительного эксперимента проводятся за секунды. В таблице 2 представлены результаты замера времени работы симулятора для входных потоков заданий от 1000 до 5000.

Таблица 2. Время проведения эксперимента в секундах с симулятором Elytra в зависимости от размера входного потока заданий

Число заданий	FCFS	Backfill
1000	0,8	1,4
2000	1,4	3,9
3000	2,1	8,3
4000	2,6	12,5
5000	3,1	18,7

5. Заключение

В качестве инструмента имитационного моделирования высокопроизводительной вычислительной системы, включающей в свой состав территориально распределенные вычислительный установки (суперкомпьютеры), должен использоваться программный симулятор, отвечающий ряду сформулированных в статье требований. В частности, симулятор должен поддерживать работу с динамическим входным потоком заданий, обеспечивать доступ к локальным очередям заданий и синхронизацию модельного времени между всеми моделируемыми суперкомпьютерами и внешним источником. Выполнение указанных требований возможно за счет реализации некоторого протокола, названного внешним интерфейсом управления симулятора системы управления суперкомпьютерными заданиями (СУЗ).

Анализ существующих средств моделирования – симуляторов СУЗ – показал, что среди них отсутствуют инструменты, удовлетворяющие выдвинутым требованиям. На основе опыта применения распространенного симулятора Alea была предложена архитектура нового симулятора системы управления заданиями, получившего название Elytra. В симуляторе Elytra реализован внешний интерфейс управления, удовлетворяющий сформулированным требованиям. Проведенные эксперименты продемонстрировали работоспособность Elytra, что позволяет сделать вывод о возможности построения на его основе адекватной модели территориально распределенной высокопроизводительной вычислительной системы.

Работа была выполнена в рамках государственного задания по теме FNEF-2024-0016.

Supercomputer Job Management System Simulator with External Control Interface

D. Lyakhovets, A. Baranov, A. Kudrin

Abstract. Simulators are popular tools for studying the supercomputer workload managers as complex multi-user systems. The paper formulates requirements for a simulator of a HPC system that includes geographically distributed supercomputers. Compliance with the stated requirements can be ensured by implementing an external simulator control interface. An analysis of the characteristics of modern HPC workload manager simulators is presented from the stated requirements point of view. The architecture of a workload manager simulator with external control interface is proposed. The first results of using the Elytra simulator, which implements the proposed architecture, are considered.

Keywords: high performance computing, simulating, job scheduling, workload manager

Литература

1. А.В. Баранов, А.И. Тихомиров. Методы и средства организации глобальной очереди заданий в территориально распределенной вычислительной системе. «Вестник ЮУрГУ. Серия: Вычислительная математика и информатика», Т. 6 (2017), № 4, 28-42.

2. А.Г. Феоктистов, А.С. Корсуков, Ю.А. Дядькин. Инструментальные средства имитационного моделирования предметно-ориентированных распределенных вычислительных систем. «Системы управления, связи и безопасности», № 4 (2016), 30–60.

3. D. Cameron, R. Carvajal-Schiano, A. Millar, C. Nicholson, K. Stockinger, F. Zini. OptorSim: A simulation tool for scheduling and replica optimisation in data grids. “Computing in High Energy and Nuclear Physics”, 2010, 707-711.

4. S. Bąk, M. Krystek, K. Kurowski, A. Oleksiak, W. Piatek, J. Waglarz. GSSIM - A tool for distributed computing experiments. “Scientific Programming”, V. 19 (2017), 231-251.

5. W. Chen, E. Deelman. WorkflowSim: A toolkit for simulating scientific workflows in distributed environments. “2012 IEEE 8th International Conference on E-Science, e-Science 2012”, 2012, 1-8.

6. S. Ostermann, K. Plankensteiner, R. Prodan, T. Fahringer. GroudSim: An Event-Based Simulation Framework for Computational Grids and Clouds. “Euro-Par 2010 Parallel Processing Workshops. Euro-Par 2010. Lecture Notes in Computer Science”, V. 6586 (2011), 305–313.

7. P.-F. Dutot, M. Mercier, M. Poquet, O. Richard. Batsim: A Realistic Language-Independent Resources and Jobs Management Systems Simulator. “Job Scheduling Strategies for Parallel Processing. Lecture Notes in Computer Science”, V. 10353 (2017), 178-197.

8. M. Obaida, J. Liu. Simulation of HPC job scheduling and large-scale parallel workloads. “2017 Winter Simulation Conference (WSC)”, 2017, 920-931.

9. D. Klusáček, M. Soysal, F. Suter. Alea – Complex Job Scheduling Simulator. “Parallel Processing and Applied Mathematics. PPAM 2019. Lecture Notes in Computer Science”, V. 12044 (2020), 217-229.

10. N. Capit, G. Da Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounié, P. Neyron, O. Richard. A batch scheduler with high level components. “CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid”, V. 2 (2005), 776-783.

11. D. Klusáček, M. Soysal. Walltime Prediction and Its Impact on Job Scheduling Performance and Predictability. “Job Scheduling Strategies for Parallel Processing. JSSPP 2020. Lecture Notes in Computer Science”, V. 12326 (2020), 127-144.

12. V. Chlumský, D. Klusáček. Improving Accuracy of Walltime Estimates in PBS Professional Using Soft Walltimes. “Job Scheduling Strategies for Parallel Processing. JSSPP 2022. Lecture Notes in Computer Science”, V. 13592 (2023), 192-210.

13. D. Lyakhovets, A. Baranov. Efficiency Thresholds of Group Based Job Scheduling in HPC Systems. “Lobachevskii Journal of Mathematics”, V. 43 (2023), 2863-2876.

14. M. Jaros, D. Klusáček, J. Jaros. Optimizing Biomedical Ultrasound Workflow Scheduling Using Cluster Simulations. “Job Scheduling Strategies for Parallel Processing. JSSPP 2020. Lecture Notes in Computer Science”, V. 12326 (2020), 68-84.

15. A. Baranov, D. Lyakhovets. Accuracy Comparison of Various Supercomputer Job Management System Models. “Accuracy Comparison of Various Supercomputer Job Management System Models”, V. 42 (2021), 2510–2519.

16. G. I. Savin, B. M. Shabanov, P. N. Telegin, and A. V. Baranov, “Joint Supercomputer center of the Russian Academy of Sciences: Present and future,” Lobachevskii J. Math. 40 (2019). 1853–1862.

17. А.В. Баранов, Д.С. Ляховец. Имитационная модель системы пакетирования суперкомпьютерных заданий на базе симулятора Alea. «Программные продукты и системы», №4 (2022), 631-643.

18. W. Cirne and F. Berman, “A model for moldable supercomputer jobs,” in Proceedings of the 15th International Parallel and Distributed Processing Symposium IPDPS 2001 (2001), p. 8.

19. D. Lyakhovets, A. Baranov, P. Telegin. Scale Ratio Tuning of Group Based Job Scheduling in HPC Systems. “Lobachevskii Journal of Mathematics”, V. 44 (2024), 5012-5026.

Современные и перспективные процессоры для высокопроизводительных вычислений

А.С. Шмелёв[1]

[1] МСЦ РАН – филиал ФГУ ФНЦ НИИСИ РАН, НИЦ «Курчатовский институт», Москва, Россия, guest8993@rambler.ru

Аннотация. В наши дни уже невозможно представить развитие науки и техники без компьютерного моделирования, проводимого на высокопроизводительных вычислительных машинах. В данной работе проводится обзор современных процессоров различных архитектур для вычислительных центров, приведены их показатели производительности, а также приведены анонсы перспективных моделей процессоров и показаны тенденции в данной области.

Ключевые слова: высокопроизводительные вычисления, центральный процессор, ускорители вычислений, суперЭВМ, TOP500, x86, POWER, ARM, управление потоком данных, ВПП

1. Введение

Современные высокопроизводительные вычислительные системы уже имеют производительность, позволившую вывести численное моделирование физических явлений и инженерных систем фактически на один уровень с натурным экспериментом.

Постоянное увеличение производительности суперкомпьютеров не является самоцелью, а позволяет повысить качество моделирования физических процессов с помощью перехода к более сложным моделям, использовать которые ранее не было возможности ввиду слишком большого времени, необходимого для проведения требуемого объёма вычислений над требуемым объёмом данных. В актуальном рейтинге среди пятисот наиболее производительных суперкомпьютеров, установленных в мире (TOP500), все позиции занимаются скалярными многопроцессорными машинами, а производительность суперкомпьютеров, занимающих верхние строчки рейтинга, перевалила за петафлоп. Наиболее часто сейчас при построении суперкомпьютеров используются многоядерные суперскалярные микропроцессоры, число вычислительных ядер в таких процессорах может достигать сотен единиц, а общее число ядер в суперкомпьютерах уже измеряется миллионами. В силу ряда причин в последние годы пиковую производительность процессоров, как и пиковую производительность суперкомпьютеров в целом, наращивали за счёт увеличения вычислительных ядер. Как правило, реальная же производительность выполнения научно-технических программ, при равной пиковой производительности, бывает у той вычислительной машины, которая содержит меньшее число, но более производительных ядер. Это объясняется тем, что на распараллеливаемых участках кода с увеличением числа процессорных ядер возрастают и накладные расходы на синхронизацию вычислений и обмен данными между процессорными ядрами, а на последовательных - время выполнения будет определяться производительностью одного ядра. Таким образом, при выборе процессора для построения суперкомпьютера для высокопроизводительных вычислений следует отдавать предпочтение вариантам с наиболее производительными ядрами. На сегодняшний день на рынке процессоров для суперЭВМ представлены чипы нескольких архитектур, далее их и рассмотрим.

2. x86

В пределах архитектуры x86 есть два основных игрока – это собственно создатели архитектуры и многолетние лидеры рынка - фирма Intel и их многолетний основной конкурент, на чьих процессорах сейчас построен самый производительный суперкомпьютер – это фирма AMD, с её продукции и начнём обзор.

В некоторых серверных процессорах Epyc, начиная с третьего поколения, используется технология 3DV-Cache, для увеличения КЭШ третьего уровня, в частности, в процессорах третьего поколения это позволило увеличить КЭШ L3 с 256 до 768 Мбайт. Благодаря наложению чиплетов и использованию сквозных кремниевых межсоединений (through-silicon vias (TSVs)) удалось добиться большей плотности, меньшей задержки и меньшего тепловыделения, чем если бы добавляли такой объём КЭШ классическим способом с размещением в одной плоскости [1]. Эти процессоры, в частности, устанавливались в суперкомпьютер Frontier, занимающий с июня 2022 года и по настоящий момент первую строчку в рейтинге top500 [2].

В сентябре 2023 года к уже имеющемуся разделению на обычные процессоры и процессоры с поддержкой большого объёма КЭШ компания AMD представила процессоры на микроархитектуре Zen4c, с энергоэффективными ядрами меньшего размера [3]. Данная микроархитектура предназначена, в том числе, и для центров обработки данных для облачных сервисов, хранилищ и т.д. [3]. Серверные процессоры с микроархитектурой четвёртого поколения Zen4 имеют до 96 ядер и суммарно до 1152 МБ КЭШ третьего уровня с поддержкой технологии 3DV-Cache [4]. Процессоры поддерживают работу с шиной PCI-Express 5-ого поколения и памятью DDR5. В частности, один из самых производительных процессоров с микроархитектурой Zen4 AMDEPYC 9684X имеет следующие характеристики: 96 ядер, работающих на базовой частоте 2,55 ГГц (любое ядро может разгоняться до 3.7ГГц, или все ядра сразу до 3.42 ГГц), 1254 МБ Кэш суммарно (L1 – 6Мбайт, L2 – 96Мбайт и L3 – 1152МБайт), 128 линий связи с PCIe и 12 каналов для работы с памятьюDDR5-4800 с поддержкой ECC, требования по теплоотводу(TDP) в 400 Вт при стоимости 14756 долларов США на момент выхода [4]. Также в процессорах четвёртого поколения была введена поддержка векторных инструкций AVX-512, правда, через 256-битную шину данных, в результате чего использование этих инструкций будет приводить либо к падению тактовой частоты, либо к увеличению энергопотребления [5].

Компания AMD уже анонсировала серверные процессоры следующего поколения на макроархитектурахZen5c с компактными ядрами[6], Zen5[6] и Zen5 с поддержкой технологии 3D-VCache[7], назначив выход на вторую половину 2024 года. Так, планируется, что серверные процессоры, нацеленные на область высокопроизводительных вычислений (HPC) будут иметь до 128 ядер Zen5 по два потока на ядро [8][9], и будет производиться с тем же сокетом SP5, что и процессоры предыдущего поколения, т.е. пользователи теоретически будут иметь возможность повысить производительность системы без капитальной её перестройки. Производить данные процессоры планируют по 4нм техпроцессу (и по 3нм техпроцессу процессоры на основе компактных ядер Zen5c).

Что касается отличия ядер архитектуры текущего поколения и следующего, то планируется полноценная поддержка векторных команд AVX-512 по 512 битной шине данных [10], введение полнофункционального блока выполнения векторных команд в состав процессора дает ему ощутимые преимущества по повышению производительности, но переход на более длинные вектора пока не предвидится, для этого понадобится увеличивать длину строки в КЭШ памяти. По заявлениям компании AMD, в новых процессорах улучшили систему предсказания переходов, увеличили окно команд, выдаваемых не по порядку, что в свою очередь позволило увеличить количество выдаваемых в такт команд, увеличили количество исполняемых устройств в ядре и количество векторных регистров, а также размеры КЭШ [10,11,12]. Т.е. как мы видим, в новой микроархитектуре ядро процессора подверглось серьёзной переработке, и увеличение производительности процессора попытались достичь не только за счёт увеличения количества ядер, как это обычно мы видим в последнее время, но упор был сделан на увеличении производительности отдельного ядра процессора.

Как было сказано в [13], для оптимального использования векторных команд все элементы вектора должны находиться в пределах одной строки КЭШ, и без этого дальнейшее увеличение аппаратной длины вектора нецелесообразно. В новых процессорах длину строки КЭШ оставили равной 64 Байтам, соответственно увеличения аппаратной длины векторов в векторных инструкциях ждать пока не приходится.

Прямым конкурентом фирмы AMD на рынке серверных процессоров для высокопроизводительных вычислений является компания Intel. Так же как и AMD, они разделили свою линейку серверных процессоров по нишам, ориентированным на облачные вычисления, с большой плотностью более простых ядер с меньшим энергопотреблением (так называемые E ядра) и процессоры, ориентированные на область высокопроизводительных вычислений (P ядра с большей частотой и производительностью)[14].

Начиная с процессоров Xeon 4-ого поколения, компания Intel перешла на построение серверных процессоров из нескольких чипов на общей подложке. Текущее пятое поколение процессоров Xeon с кодовым названием Emerald Rapids было запущено в производство в декабре 2023 года, производится с использованием 7нм техпроцесса и имеет тот же сокет, что и процессоры предыдущего поколения. Топовый процессор этого поколения Intel Xeon Platinum 8593Q имеет 64 ядра (128 потоков), работающих на базовой частоте 2,2 ГГц (с максимальным разгоном частоты ядра до 3.9 ГГц, либо всех ядер сразу до 3ГГц), общий размер объединённого смарт КЭШ составляет 320МБ, процессор поддерживает работу с памятью DDR-5600 с поддержкой ECC и имеет требования к теплоотводу 385 Вт. Рекомендуемая стоимость такого процессора составляет 12400 долларов США [15].

В 2024 году компания Intel планирует выпустить на рынок свои серверные процессоры Xeon следующего, 6-ого поколения с кодовым названием Granite Rapids.

Процессор содержит несколько модулей, называемых вычислительными тайлами (Compute tile), каждый такой модуль содержит до 44 P-ядер, кэш память и контроллеры внешней памяти DDR5. Кэш память первого уровня увеличили до 112 КБ на ядро (вдвое увеличили кэш команд до 64КБ и оставшиеся 48 КБ на КЭШ данных). При этом на каждое ядро приходится по 2МБ кэш второго уровня и 4МБ кэш третьего уровня (суммарно 512МБ КЭШ L3)[16]. Также был улучшен модуль предсказания переходов, добавлена аппаратная поддержка шифрования AES-256 и переработан модуль AMX (Advanced Matrix Extensions), в него добавлена поддержка данных типа FP16, что, как планируют, приведёт к ускорению приложений, связанных с машинным обучением. Планируется, что процессоры, ориентированные на область высокопроизводительных вычислений будут иметь до 128 двухпотоковых ядер, 12 каналов к памяти DDR5 (заявлена поддержка MCRDIMM) и до 136 линий PCI Express 5.0.

Суперкомпьютер Aurora, занимающий сейчас вторую строчку в списке пятисот самых производительных машин в мире [17], был построен с использованием процессоров серии Intel Xeon Max, характеризующихся возможностью работы с памятью HBM. В грядущем поколении процессоров поддержка HBM не заявлена, вместо этого в процессорах Xeon 6-ого поколения заявлена поддержка технологии MCRDIMM (Multiplexer Combined Ranks DIMM) [18]. В данных модулях с помощью добавления специального буфера данных ведётся параллельная работа с двумя рангами памяти, т.е. производится вычитывание сразу 128 байт данных. Такие модули были разработаны для увеличения объёма и увеличения пропускной способности к памяти [19], увеличение объёма памяти с помощью стандартных модулей DDR5 DIMM столкнулось уже с той проблемой, что сервера банально не помещаются в стандартную 19 дюймовую стойку. Надо отметить, что компания AMD совместно с Google, Microsoft и JEDEC проталкивала другую аналогичную технологию - MRDIMM (multi-ranked buffered DIMM)[20], в частности, данные модули выпускаются фирмой Micron [21] и они, как заявляется, полностью совместимы с Intel MCRDIMM. При этом, так как JEDEC ещё не опубликовала стандарт MRDIM, ему модули не соответствуют, такое объяснение дано на сайте производителя [22], что позволяет предположить идентичность, или, как минимум, сильную схожесть технологий. Так что в серверах ближайшей пары лет будут использоваться модули памяти с одновременным доступом к нескольким рангам.

3. POWER

На девятой позиции летнего списка top500 2024 года стоит суперкомпьютер Summit (занимал в 2018-2019 годах первую позицию списка) [23]. Суперкомпьютер построен на процессорах IBM POWER9, имеющих 22 ядра и работающих на частоте 3,07 ГГц. В относительно недавнем прошлом суперкомпьютеры на основе процессоров архитектуры IBM POWER были достаточно широко представлены в списке самых производительных систем, сейчас же их осталось всего 6 штук (на процессорах IBM POWER9). Процессоры текущего актуального поколения данной архитектуры POWER10 производятся компанией Samsung с использованием 7нм техпроцесса и содержат 18млрд транзисторов при площади 602мм квадратных (вариант с одним процессорным чипом в корпусе). Существует два варианта процессорных чипов, одни имеют до 15 ядер с поддержкой до восьми потоков (SMT8), другие до 30 ядер с поддержкой до 4 потоков (SMT4). Каждое ядро SMT8 содержит 80КБ кэш первого уровня (48КБ кэш команд и 32КБ кэш данных), 2МБ кэш второго уровня [24], всего на процессор приходится до 120МБ кэш третьего уровня. На самом деле в процессорных чипах реализованы 16 ядер и 128мб кэш третьего уровня, но пользователю доступно только 15 ядер и 120МБ, IBM пошла на такое в целях повышения выхода годных кристаллов [25]. Процессоры выпускаются в корпусах с одним (SCM - Single-Chip Module) или двумя (DCM – Dual-Chip Module) процессорными чипами (до 24 ядер STM8 работающих на частоте 2.95-4ГГЦ, в зависимости от варианта). До 16 SCM или до 4 DCM процессоров могут быть объединены в систему с общей памятью через интерфейс PowerAXON. POWER10 может работать с различными типами памяти, как то: DDR3-DDR5, GDDR, HBM и энергонезависимой памятью ёмкостью до 2 ПБ, в процессоре имеется два восьмиканальных контроллера памяти. Несмотря на то, что в текущих рейтингах суперкомпьютеров отсутствуют машины, построенные на процессорах последнего поколения архитектуры POWER, и следующее поколение пока не анонсировано, хоронить данную архитектуру пока рано. В 2021 году IBM продемонстрировала чип, изготовленный по 2нм технологии, и планирует использовать свой опыт и наработки для создания процессоров в будущем [26].

4. ARM

Наряду с уже традиционными в HPC процессорными архитектурами x86 и POWER последнее время получили распространение процессоры с архитектурой ARM. Например, в 2020 году первую строчку рейтинга TOP500 занимал суперкомпьютер Fugaku [27], построенный на процессорах A64FX фирмы Fujitsu. Суперскалярный процессор A64FX производится по 7нм техпроцессу и использует систему команд ARM v8.2-A с векторным расширением SVE (Scalable Vector Extension) с длиной вектора 512 бит, он был разработан специально под высокопроизводительные вычисления. Он имеет 52 ядра, 48 из которых вычислительные и 4 вспомогательные, работающих на частотах до 2.2 ГГц, общая пиковая производительность процессора до 13,5ТФлопс. Кэш память первого уровня имеет объём в 6Мбайт, по 3МБайт для команд и данных (по 64Кбайт для команд и данных в каждом ядре), 32МБайта приходится на кэш второго уровня (4 NUMA блока по 12 ядер, 8Мбайт на блок), длина строки кэш памяти 256 байт. Процессор имеет четыре контроллера памяти, поддерживающих работу с памятью HBM2(4 стека) ёмкостью до 32Гбайт, и обеспечивающих общую пропускную способность подсистемы памяти 1024 Гбайт/с. Процессор поддерживает сеть TofuD и имеет 16 линий PCIe Gen3. Сейчас в рейтинге присутствует девять машин на этом процессоре, в предыдущих рейтингах встречался ещё один ARM процессор – Marvell Thunder X2, планировалось развитие этой процессорной линейки, были образцы процессора Thunder X3, и велась разработка Thunder X4 [29], но в 2020 году проект фактически закрылся с уходом одного из ведущих сотрудников из компании.

В будущем компания Fujitsu планирует продолжать разработку серверных процессоров с архитектурой ARM, уже анонсирован процессор следующего поколения с названием Monaka. Планируемый к выпуску в 2027 году процессор будет иметь 144 ядра с системой команд ARM v9.0-A с векторным расширением SVE2, 12 каналов к памяти DDR5, а также заявлена поддержка PCI Express Gen 6 (CXL3.0) [30]. Процессор будет иметь воздушное охлаждение. В отличие от A64FX, заточенного сугубо под область HPC, процессор следующего поколения планируют сделать более универсальным.

Ещё одним немаловажным игроком на рынке высокопроизводительных вычислений является компания NVidia. В компании не стали разрабатывать свой универсальный процессор с нуля, а сделали ставку на ARM архитектуру. Компания в 2022 году анонсировала, и в первой половине 2023 года на фабрике TSMC начался выпуск микропроцессора Grace Superchip (техпроцесс 4нм). Один кристалл Grace содержит 72 ядра ARM Neoverse V2 с системой команд ARM v9.0-A (в Grace Superchip два таких кристалла). На каждое ядро на кристалле приходится по 128КБайкт кэш первого уровня (64Кбайт для команд и 64 Кбайт для данных) и 1Мбайт кэш L2, кэш L3 общий для всех ядер кристалла и имеет ёмкость 117 Мбайт [31]. В Grace Superchip два кристалла Grace объединены посредством шины NVLink-C2C (900Гбайт/с) в одну систему, пиковая производительность которой составляет 7,1 Тфлопс при конструктивных требованиях по теплоотводу (TDP) в 500 Вт. Процессор поддерживает работу с памятью LPDDR5x общим объёмом до 960 Гбайт с пропускной способностью 1Тбайт/с (по 500Гбайт/с на кристалл) и имеет 128 линий PCIe Gen 5.

Как было сказано в [32], в качестве основного элемента построения суперкомпьютеров компания сделала ставку на комбинированные вычислительные узлы из процессоров общего назначения Grace и ускорителей Hopper (H100) с кодовым названием Grace Hopper (GH200). В рейтинге TOP500 уже присутствуют 7 машин, построенных на этих чипах Nvidia. Первый европейский суперкомпьютер экзафлопсного класса с названием Jupiter, постройку которого планируют завершить в 2024 году, будет построен как раз на вычислительных узлах Grace Hopper [33].

5. Dataflow

Как мы видим, все существующие процессоры для области HPC имеют архитектуру управления потоком команд, где команды выдаются на выполнение по порядку, задаваемому счётчиком команд. Для повышения производительности в таких процессорах используется конвейер команд и суперскалярный режим, что, в свою очередь, порождает конфликты информационной зависимости по данным, выражающиеся в том, что невозможно прочитать из памяти необходимый операнд, так как он ещё не записан одной из предыдущих команд. Ещё одно серьёзное ограничение порождается конфликтами по управлению, возникающими при конвейеризации команд условного перехода. Для обхода этих ограничений используют сложную аппаратуру предсказания переходов и систему выполнения команд не по порядку (out-of-order execution), но и это не позволяет выдавать на выполнение одновременно более 4-6 команд в такт из окна в несколько сотен команд. Альтернативой мог бы послужить процессор с архитектурой управления потоком данных, потенциально позволяющий получить более высокую реальную производи-

тельность на распараллеливаемых задачах. Программа для такого процессора представляет собой ориентированный граф, где узлы – команды, а по дугам передаются токены с данными. На исполнение может быть выдана любая команда в графе при условии готовности всех её операндов, после использования токенов-операндов они уничтожаются, а по завершении выполнения команды формируется новый токен с результатом, одновременно являющимся операндом для какой-то другой команды, согласно графу, т.е. действует принцип однократного присваивания. В таком процессоре отсутствует центральное устройство управления в виде счётчика команд, а готовые к выполнению команды ищутся в распределённой на множество модулей ассоциативной памяти поиска пар (ППП) по готовности всех их операндов. В процессоре с управлением потоком данных можно одновременно исполнять команды из нескольких программ, у него отсутствуют накладные расходы на переключение между потоками, и, в теории, можно одновременно выдать на выполнение все независимые по данным команды.

Несмотря на все эти преимущества до сих пор не было сделано ни одного коммерчески успешного универсального процессора с подобной архитектурой. В первую очередь из-за сложностей реализации одного из основных узлов – быстродействующей ППП достаточной ёмкости. Дело в том, что ППП, где хранятся все не парные токены, должна обладать достаточной ёмкостью, так как переполнение её приводит к остановке процессора, и достаточным быстродействием, так как все токены проходят кольцо обработки данных (ППП-коммутатор-память, команда-исполнительное устройство и обратно через коммутатор в один из модулей ППП), и, в том числе её быстродействие определяет путь по кольцу в тактах и, соответственно, напрямую влияет на производительность на цепочках команд с зависимостью по данным.

В МСЦ РАН разрабатывалась VHDL модель векторного потокового процессора (ВПП). За счёт введения команд векторной обработки данных с хранением элементов векторов в отдельной линейно адресуемой памяти векторов(ПВ) удалось добиться существенного снижения требований к ёмкости ППП, а хранение больших массивов данных в виде векторов-указателей (вектор, элементами которого являются вектора) вместе со специальными векторными командами для сборки и рассыпания векторов на элементы позволило повысить степень векторизации программ.

Результаты моделирования показали, что на хорошо распараллеливаемых вычислительных задачах, таких как перемножение матриц, может быть достигнута высокая реальная производительность как на одном процессорном ядре, так и почти линейное увеличение производительности (1.97) было достигнуто на модели с двумя ядрами, при этом на одном из ядер исполнялась и управляющая часть программы, что достигалось за счёт малых накладных расходов на переключение между блоками перемножения подматриц в блочном алгоритме программы [34].

Моделирование задачи шаблонных вычислений (Stencil computations) показало, что на ВПП можно получить более высокую реальную производительность относительно процессоров традиционной архитектуры, несмотря на ограничивающую производительность пропускную способность к внешней ПВ(ВПВ), а с увеличением количества точек в шаблоне, в отличие от традиционных процессоров, растёт и локальность обращений в ПВ, что ещё больше увеличивает разрыв по производительности с процессорами традиционной архитектуры [34, 35].

Одно из основных преимуществ ВПП – возможность поиска готовых к выполнению команд практически во всей программе сразу, что позволяет использовать мелкогранулярный параллелизм и получать высокую производительность не только на хорошо распараллеливаемых задачах начиная с малых размерностей массивов, но и, в некоторых случаях, распараллеливать и частично векторизовать скалярный код с зависимостью по данным, что было показано на примере программы пузырьковой сортировки [36]. На распараллеливаемых алгоритмах сортировки, таких как быстрая сортировка (Quicksort) и битонная сортировка (Bitonic sort) ВПП так же показывал более высокую производительность в сравнении с фон-неймановскими процессорами [37, 38].

Результаты моделирования типовых задач позволяют утверждать, что ВПП может иметь преимущество не только на хорошо векторизуемых задачах, но и на задачах с мелкоструктурным и нерегулярным параллелизмом.

Выделение и высвобождение памяти под вектор в ВПП происходит аппаратно, блоками фиксированной длины равной максимальной аппаратной длине вектора (в модели VLmax=256 элементов). Адресом вектора является адрес первого его элемента, и вместе с длиной вектора он составляет аппаратный указатель на вектор и передаётся в поле данных токена от команды к команде, и программист к нему доступа не имеет. Т.е. программист не может заранее знать точно, по какому физическому адресу будут записаны элементы вектора в ПВ, поэтому извне заранее загрузить ПВ процессора невозможно, но возможно получить ссылку из процессора на область памяти, в которую потом можно записать

подготавливаемые данные при наличии какого-то внешнего арбитра ПВ, если использовать ВПП в качестве ускорителя. Для загрузки программы в память команд процессора предусмотрено два канала, отдельно для скалярных команд и векторных команд. Интерфейс также предусматривает загрузку извне в процессор как отдельных токенов по выделенному каналу с передачей на скалярные команды, так и шину для передачи элементов вектора для подачи на команду формирования вектора (ФВ). Выдача токенов также возможна как одиночного токена через отдельный канал, так и через отдельный канал потока токенов. В целом, ВПП возможно использовать в качестве ускорителя, но хотелось бы иметь возможность предварительной подготовки данных для обработки в процессоре, этот момент требует дальнейшей проработки.

6. Заключение

В серверных процессорах уже давно стандартом де-факто стало наличие векторных расширений команд, и на сегодняшний день практически все серверные процессоры имеют блоки выполнения векторных команд. Длина векторов пока невелика, 2-8 64-битных слов, и ограничивается прежде всего двумя параметрами: длиной строки кэш, так как для достижения высокой производительности на векторной обработке все элементы вектора должны помещаться в одной строке, и сбалансированностью векторной и скалярной составляющей, так как при слишком большом перевесе в производительности векторной части выполнение скалярного кода на не векторизуемых участках программы будет превосходить время выполнения векторных команд. Как мы видим, в процессорах последних поколений производители делают упор на повышение производительности скалярной обработки, улучшая схемы предсказаний переходов и окно выдачи команд не по порядку, что позволяет увеличить количество выдаваемых на исполнение команд в такт.

В общем, наметилось разделение серверных процессоров по различным нишам: HPC и задачи машинного обучения. В стане x86 производители-гиганты выпускают различные модели процессоров (собственно, NVidia тоже разделила свои линейки ускорителей), более мелкие производители, не имеющие возможность выпускать разнонаправленные продукты, концентрируются на чём-то одном (например, Fujitsu на рынке HPC, Ampere на рынке облачных сервисов и машинного обучения). Тем не менее, во все современные процессоры добавляется поддержка FP16 и INT8 как раз для повышения производительности на задачах машинного обучения.

Одной из возможных альтернатив существующим процессорам в перспективе могут стать процессоры с архитектурой управления потоком данных, потенциально позволяющие получить более высокую реальную производительность на распараллеливаемых задачах.

К сожалению, ввиду общей обстановки в мире, рейтинг TOP500 теряет свою актуальность, так как часть высокопроизводительных систем, в частности китайских, там не представлена, таким образом, часть технических решений может остаться за кадром [39].

Работа была выполнена в рамках государственного задания по теме FNEF-2024-0016.

Review Of Modern And Upcoming Processors For High-Performance Computing

A.S. Shmelev

Abstract. Nowadays, it is impossible to imagine the progress in science and technology without computer modeling on high-performance computing machines. This paper provides an overview of modern processors of various architectures for HPC.

Keywords: High performance computing (HPC), Central processing unit (CPU), Accelerated Processing Unit (APU), supercomputer, TOP500, x86. POWER, ARM, dataflow, vector dataflow processor

Литература

1. https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/product-briefs/3d-vcache.pdf (дата обращения 06.11.2024).

2. https://top500.org/system/180047/(дата обращения 06.11.2024).
3. Пресс-релиз https://www.amd.com/en/newsroom/press-releases/2023-9-18-amd-completes-4th-gen-epyc-family-with-the-amd-epy.html(дата обращения 06.11.2024).
4. https://www.amd.com/en/products/processors/server/epyc/4th-generation-9004-and-8004-series/amd-epyc-9684x.html (дата обращения 06.11.2024).
5. AMD EPYC™ 9004 Series Architecture Overview June, 2023 https://www.amd.com/content/dam/amd/en/documents/epyc-technical-docs/white-papers/58015-epyc-9004-tg-architecture-overview.pdf (дата обращения06.11.2024).
6. AMD announces 3nm EPYC Turin with 192 cores and 384 threads — 5.4X faster than Intel Xeon in AI work, launches second half of 2024 News By Paul Alcorn published June 3, 2024 https://www.tomshardware.com/pc-components/cpus/amd-announces-3nm-epyc-turin-launching-with-192-cores-and-384-threads-in-second-half-of-2024-54x-faster-than-intel-xeon-in-ai-workload (дата обращения 06.11.2024).
7. AMD FAD 2022 AMD CPU Core Roadmap To Zen 5https://www.servethehome.com/amd-technology-roadmap-from-amd-financial-analyst-day-2022/amd-fad-2022-amd-cpu-core-roadmap-to-zen-5/(датаобращения 06.11.2024).
8. https://3dnews.ru/1105849/amd-podelilas-nekotorimi-podrobnostyami-o-servernih-protsessorah-turin-i-turin-dense-s-arhitekturami-zen-5-i-zen-5c (дата обращения 06.11.2024).
9. https://www.tomshardware.com/pc-components/cpus/amd-announces-3nm-epyc-turin-launching-with-192-cores-and-384-threads-in-second-half-of-2024-54x-faster-than-intel-xeon-in-ai-workload (дата обращения 06.11.2024).
10. https://www.phoronix.com/review/amd-zen5-ryzen-9000 (дата обращения 06.11.2024).
11. Zen 5's 2-Ahead Branch Predictor Unit: How a 30 Year Old Idea Allows for New Tricks July 26, 2024 Cheese, Camachohttps://chipsandcheese.com/2024/07/26/zen-5s-2-ahead-branch-predictor-unit-how-30-year-old-idea-allows-for-new-tricks/ (дата обращения 06.11.2024).
12. https://chipsandcheese.com/2024/07/15/a-video-interview-with-mike-clark-chief-architect-of-zen-at-amd/ (дата обращения 06.11.2024).
13. Дикарев Н.И., Шабанов Б.М. Выбор процессора для СуперЭВМ и развитие межведомственного суперкомпьютерного центра РАН // Многопроцессорные вычислительные и управляющие системы – 2007. Материалы Международной научно-технической конференции. Т.1. – Таганрог: изд-во ТТИ ЮФУ, 2007.
14. https://www.anandtech.com/show/17259/intel-discloses-multigeneration-xeon-scalable-roadmap-new-ecore-only-xeons-in-2024 (дата обращения 06.11.2024).
15. https://ark.intel.com/content/www/us/en/ark/products/237252/intel-xeon-platinum-8593q-processor-320m-cache-2-2-ghz.html.
16. https://www.nextplatform.com/2023/09/22/intel-gets-its-chiplets-in-order-with-5th-gen-xeon-sps/(дата обращения 06.11.2024).
17. https://top500.org/system/180183/ (дата обращения 06.11.2024).
18. https://www.hardwareluxx.ru/index.php/news/hardware/prozessoren/55643-intel-xeon-6-144-effektivnykh-yadra-na-starte.html (дата обращения 06.11.2024).
19. https://www.servethehome.com/what-is-a-mcr-dimm-or-multiplexer-combined-ranks-dimm-sk-hynix-micron-samsung-intel-amd-nvidia/ (дата обращения 06.11.2024).
20. https://www.tomshardware.com/news/amd-advocates-ddr5-mrdimms-with-speeds-up-to-17600-mts(дата обращения 06.11.2024).
21. https://www.micron.com/content/dam/micron/global/public/documents/products/product-flyer/mrdimm_product_brief.pdf (дата обращения 06.11.2024).
22. https://www.micron.com/products/memory/dram-modules/mrdimm#accordion-9331643c5c-item-afcfffb36e (дата обращения 06.11.2024).
23. https://top500.org/system/179397/ (дата обращения 06.11.2024).
24. https://www.hardwareluxx.de/index.php/news/hardware/prozessoren/53864-ibm-power10-bietet-30-kerne-mit-smt8-pcie-5-0-und-ddr5.html (дата обращения 06.11.2024).
25. https://www.servethehome.com/ibm-power10-searching-for-the-holy-grail-of-compute/ (дата обращения 06.11.2024).
26. https://www.anandtech.com/show/16656/ibm-creates-first-2nm-chip (дата обращения 06.11.2024).
27. https://top500.org/system/179807/ (дата обращения 06.11.2024).
28. https://www.fujitsu.com/downloads/JP/jsuper/a64fx/a64fx_datasheet.pdf (дата обращения 06.11.2024).

29. https://www.anandtech.com/show/15621/marvell-announces-thunderx3-96-cores-384-thread-3rd-gen-arm-server-processor (дата обращения 06.11.2024).
30. Презентация Next Arm Processor FUJITSU-MONAKA and Its Technologies, Fujitsu 2024, https://www.fujitsu.com/global/imagesgig5/FUJITSU-MONAKA.pdf.
31. https://developer.nvidia.com/blog/nvidia-grace-cpu-superchip-architecture-in-depth/ (дата обращения 06.11.2024).
32. Шмелёв А.С. Тенденции в графических ускорителях для высокопроизводительных вычислений // Труды научно-исследовательского института системных исследований Российской академии наук. 2023. Т. 13. № 4. С. 117-122.
33. https://www.fz-juelich.de/en/ias/jsc/jupiter (датаобращения06.11.2024).
34. Дикарев Н.И., Шабанов Б.М., Шмелёв А.С. Моделирование параллельной работы ядер векторного потокового процессора с общей памятью // Программные системы: теория и приложения. 2018. Т. 9. № 1 (36). С. 37-52.
35. Дикарев Н.И., Шабанов Б.М., Шмелёв А.С. Недостаточная пропускная способность памяти на программе Stencil: преимущество векторного потокового процессора // Программные системы: теория и приложения. 2018. Т. 9. № 4 (39). С. 399-415.
36. Дикарев Н.И., Шабанов Б.М., Шмелёв А.С. Мелкоструктурный параллелизм и более высокая производительность процессорного ядра: преимущества векторного потокового процессора // Программные системы: теория и приложения. 2019. Т. 10. № 4 (43). С. 201-217.
37. Дикарев Н.И., Шабанов Б.М., Шмелёв А.С. Быстрые алгоритмы сортировки для векторного потокового процессора // В сборнике: Суперкомпьютерные технологии (СКТ-2018). Материалы 5-й Всероссийской научно-технической конференции. 2018. С. 87-91.
38. Дикарев Н.И., Шабанов Б.М., Шмелёв А.С. Преимущества и недостатки использования метода векторов указателей в векторном потоковом процессоре // Программные системы: теория и приложения. 2021. Т. 12. № 4 (51). С. 65-83.
39. https://www.tomshardware.com/news/industry-expert-chinas-supercomputer-might-may-be-unmatched.

О квазипериодических, но не периодических элементах специального класса функциональных непрерывных дробей

М.М. Петрунин[1]

[1]НИЦ «Курчатовский институт» – НИИСИ, Москва, Россия, petrushkin@yandex.ru

Аннотация. В работе М.М. Петрунин "Об одном классе периодических элементов гиперэллиптических полей", Чебышевский сборник. 2024. Т. 25. № 4 в случае произвольной нечетной степени многочлена f над произвольным полем алгебраических чисел $\mathbb{K}$ был получен класс всегда квазипериодических в $\mathbb{K}((x))$ элементов и выделен подкласс всегда периодических элементов вида $\frac{v+w\sqrt{f}}{u}$, задаваемые только соотношениями на многочлены $u,v,w,f\in\mathbb{K}[x]$ и их степени. Класс не пуст при наличии в гиперэллиптическом поле хотя бы одного квазипериодического элемента. В настоящей работе построены новые примеры периодических, а также построены примеры квазипериодических, но не периодических элементов вышеуказанного класса квазипериодических элементов.

Ключевые слова: гиперэллиптическое поле, непрерывные дроби, функциональные непрерывные дроби, S -единицы, периодичность, квазипериодичность

1. Введение

Исследование проблемы периодичности функциональных непрерывных дробей в гиперэллиптических полях было начато в работах Абеля [2] и Чебышёва [3]. В XX и XXI веках классические результаты получили современное изложение, и для гиперэллиптических полей, заданных многочленами чётной степени, была развита теория функциональных непрерывных дробей в $\mathbb{K}((1/x))$ (см. например, [4], [5], [6], [7]). Основы для изучения непрерывных дробей в $\mathbb{K}((x))$ были заложены в работе [8], в которой впервые глубокие результаты по проблеме периодичности функциональных непрерывных дробей были получены для гиперэллиптических полей, определяемых многочленами нечётной степени. Дальнейшее развитие этот случай получил в работах [9, 10, 11, 12], где, в частности, были получены результаты, связанные с элементом $\sqrt{f}$. В связи с тем, что это лишь один из элементов гиперэллиптического поля, возникает естественный вопрос: насколько широк класс квазипериодических и периодических элементов. В. П. Платонов предложил исследовать его в случае, когда гиперэллиптическое поле порождается многочленом нечётной степени.

В случае $\mathbb{K}((1/x))$ гиперэллиптического поля, порожденного многочленом четной степени, в работе [13] был получен класс квазипериодичных элементов. Доказательство было проведено с рассмотрением последовательности дивизоров нулей и полюсов полных частных разложения в непрерывную дробь квадратичной иррациональности.

В работе [1] было получено альтернативное компактное доказательство аналогичного результата для $\mathbb{K}((x))$ в случае многочлена нечетной степени, а также впервые выделен подкласс периодических элементов.

Теорема ([1]). *Пусть $f\in\mathbb{K}[x]$ — бесквадратный многочлен степени $2g+1$. Если в поле $L=\mathbb{K}(x)(\sqrt{f})$, есть квазипериодические в $\mathbb{K}((x))$ элементы, то квазипериодичны элементы α, представимые в виде несократимой дроби $\alpha=\frac{v+x^i\sqrt{f}}{x^j r}$, $r\mid v^2-x^{2i}f$, $i,j\in\mathbb{Z}_{\ge 0}$, $\deg v\le\sigma+i$, $\deg r=\sigma+i-j$, $j-i\le\sigma$, где $\sigma=g$ или $\sigma=g+1$.*

Более того, если $v=0$, $x\nmid r$ и $\deg r\ne\sigma$, или $r\in\mathbb{K}$ или $x^{t+j}r=v^2-x^{2i}f$, то такие элементы периодичны. В последнем случае $t=i+g$ или $t=i+g+1$, и или $i=0$ или $j=0$.

Критерий квазипериодичности (см. работы [6, 10]) накладывает жесткие условия на элементы гиперэллиптического поля. В силу этого, квазипериодичность и тем более периодичность — редкие явления. При наличии в гиперэллиптическом поле хотя бы одного квазипериодического элемента теорема позволяет выделить класс всегда квазипериодичных элементов. Более того, она постулирует периодичность широкого подкласса. Однако, как показано в работе [1], если элементы класса, выделенного в теореме, содержат в разложении многочлена $v^2 - x^{2i} f$ множители отличные от r или степени x, то они могут быть квазипериодичны, но не периодичны. В настоящей работе впервые построены примеры, как квазипериодических, но не периодических, так и периодических элементов класса квазипериодических элементов, выделенного в теореме, в случае $v = 0$.

2. Примеры

Рассмотрим случай, когда $v = 0$, но $deg\, r = g + 1$.

Пример 1. *Пусть* $\alpha = \frac{\sqrt{f}}{r}$, *где*

$$f = \left(2x^2 - 2x - 1\right)\left(3x + 1\right)\left(2x + 1\right)\left(x - 1\right), \quad r = \left(2x^2 - 2x - 1\right)\left(x - 1\right).$$

Тогда α *обладает квазипериодическим, но не периодическим разложением в непрерывную дробь:*

$$\alpha = [1, \frac{1}{2x} - \frac{1}{4}, \overline{-\frac{8}{3x} - 4, \frac{3}{8x} + \frac{9}{16}, -\frac{32}{9x} - \frac{16}{9}, \frac{9}{32x} + \frac{9}{64}}^{\frac{16}{9}}].$$

Длина квазипериода равна 4, *а коэффициент квазипериодичности равен* $16/9$.

В то же время, при сохранении многочлена f, но взять в качестве r множитель степени $deg\, r = g$, то разложение уже будет периодичным.

Пример 2. *Пусть* $\alpha = \frac{\sqrt{f}}{r}$, *где*

$$f = \left(2x^2 - 2x - 1\right)\left(3x + 1\right)\left(2x + 1\right)\left(x - 1\right), \quad r = 2x^2 - 2x - 1.$$

Тогда α *обладает периодическим разложением в непрерывную дробь с длиной периода* 6*:*

$$\alpha = [-1, -\frac{1}{x} - 1, \overline{-\frac{2}{3x} - \frac{2}{3x^2} + 2, -\frac{1}{x} - \frac{3}{2}, \frac{4}{x} + 6}^{-\frac{1}{4}}]$$

Длина квазипериода равна 3, *а коэффициент квазипериодичности равен* $-1/4$.

Однако степень множителя f не является определяющим фактором, и для фиксированного f при степенях множителя $deg r = g$ и $deg r = g + 1$ могут встречаться все возможности. Как непериодичность:

Пример 3. *Пусть* $\alpha = \frac{\sqrt{f}}{r}$, *где*

$$f = \left(2x^2 - 2x - 1\right)\left(3x + 1\right)\left(2x + 1\right)\left(x - 1\right), \quad r = \left(2x + 1\right)\left(x - 1\right).$$

Тогда α *обладает квазипериодическим, но не периодическим разложением в непрерывную дробь:*

$$\alpha = [-1, -\frac{1}{2x} - \frac{1}{4}, \overline{\frac{8}{3x} + \frac{4}{3}, -\frac{3}{8x} - \frac{3}{16}, \frac{32}{9x} + \frac{16}{3}, -\frac{9}{32x} - \frac{27}{64}}^{\frac{16}{9}}].$$

Длина квазипериода равна 4, *а коэффициент квазипериодичности равен* $16/9$.

Так и периодичность:

Пример 4. *Пусть* $\alpha = \frac{\sqrt{f}}{r}$, *где*

$$f = \left(2x^2 - 2x - 1\right)\left(3x+1\right)\left(2x+1\right)\left(x-1\right), \quad r = \left(2x^2 - 2x - 1\right)\left(2x+1\right).$$

Тогда α *обладает периодическим разложением в непрерывную дробь с длиной периода 6:*

$$\alpha = [-1, \frac{1}{x}+1, \overline{-\frac{2}{x} - \frac{2}{3x^2} + \frac{2}{3}, \frac{1}{x} + \frac{1}{2}, -\frac{4}{x} - 2}^{-\frac{1}{4}}].$$

Длина квазипериода равна 3, а коэффициент квазипериодичности равен - 1/4.

3. Заключение

Работа выполнена в рамках государственного задания по проведению фундаментальных научных исследований, проект FNEF-2024-0001.

On Quasiperiodic but Nonperiodic Elements of a Special Class of Functional Continued Fractions

M.M Petrunin

Abstract. In the paper [1] for an arbitrary odd-degree polynomial f over an arbitrary field of algebraic numbers $\mathbb{K}$, a class of elements in $\mathbb{K}((x))$ was identified that are always quasiperiodic, and a subclass of always periodic elements of the form $\frac{v + w\sqrt{f}}{u}$ was distinguished. These are defined solely by relations on the polynomials $u, v, w, f \in \mathbb{K}[x]$ and their degrees. This class is non-empty if at least one quasiperiodic element exists in the hyperelliptic field. In the present work, new examples of periodic elements are constructed, as well as examples of quasiperiodic but non-periodic elements within the aforementioned class of quasiperiodic elements.

Keywords: hyperelliptic field, continued fractions, functional continued fractions, S-units, periodicity, quasiperiodicity

Литература

1. Петрунин М. М. Об одном классе периодических элементов гиперэллиптических полей. «Чебышевский сборник». 2024, Т. 25, № 4.

2. Abel N.H. Ueber die Integration der Differential-Formel $\rho dx / \sqrt{R}$ wenn R und ρ ganze Functionen sind. «Journal für die reine und angewandte Mathematik». 1826, Vol. 1, P. 185–221.

3. Tchebicheff P. Sur l'intégration des différentielles qui contiennent une racine carrée d'un polynome du troisieme ou du quatrieme degré'. «Journal des math. pures et appl.» 1857, Vol. 2, P. 168–192.

4. Платонов В. П. Теоретико-числовые свойства гиперэллиптических полей и проблема кручения в якобианах гиперэллиптических кривых над полем рациональных чисел. «Успехи Математических Наук». 2014, Т. 69:1, № 415, С. 3–38.

5. Adams William W., Razar Michael J. Multiples of point on elliptic curves and continued fractions. «Proc. London Math. Soc.». 1980, Vol. 41, no. 3. P. 481–498.

6. Schmidt Wolfgang M. On continued fractions and Diophantine approximation in power series fields. «Acta arithmetica». 2000, Vol. 95, no. 2, P. 139–166.

7. Schinzel A. On some problems of the arithmetical theory of continued fractions. «Acta Arithmetica». 1961, Vol. 6, №. 4, P. 393-413.

8. Беняш-Кривец В. В., Платонов В. П. Группы S-единиц в гиперэллиптических полях и непрерывные дроби. «Математический сборник». 2009. Т. 200, № 11. С. 15–44.

9. Петрунин М. М. S-единицы и периодичность квадратного корня в гиперэллиптических полях. «Доклады Академии наук». 2017. Т. 474, №. 2, – С. 155-158.

10. Платонов В. П., Петрунин М. М. Группы S-единиц и проблема периодичности непрерывных дробей в гиперэллиптических полях. «Тр. МИАН». 2018.

11. Платонов В. П., Федоров Г. В. О проблеме периодичности непрерывных дробей в гиперэллиптических полях. «Математический сборник». 2018. Т. 209, № 4, С. 54–94.

12. Платонов В. П. Об описании периодических элементов эллиптических полей, заданных многочленом третьей степени. «Успехи Математических Наук». 2024, Т. 79, В. 6.

13. Berry T.G. On periodicity of continued fractions in hyperelliptic function fields. «Arch. Math. (Basel)». 1990, Vol. 55, no. 3, P. 259–266.

О теореме Успенского-Райса

А. И. Грюнталь[1]

[1]НИЦ «Курчатовский институт» – НИИСИ, Москва, Россия, grntl@niisi.ras.ru

Аннотация. В статье содержится подробное доказательство теоремы Успенского-Райса. Вводятся основные понятия, относящиеся к вычислимым функциям. Используется вычислительная модель «Машина с неограниченными регистрами». Изложение носит замкнутый и элементарный характер.

Ключевые слова: вычислимые функции, программа, машина с неограниченными регистрами

1. Введение

Предлагаемая статья содержит подробное (как автор надеется) доказательство теоремы Успенского-Райса. Эта теорема имеет важное общенаучное значение. Понимание её смысла и доказательства представляет собой существенный элемент образования программистов. Поэтому полное и простое изложение доказательства этой теоремы представляется целесообразным.

Теорема Успенского-Райса – это математическое утверждение, и поэтому все понятия, которые применяются в формулировке теоремы или используются при доказательстве, должны быть математически определены.

Однако само понятие программы точным не является. Например, стандарт языка программирования C содержит несколько тысяч страниц и не подлежит формализации.

Можно вместо математически точных понятий использовать их аналоги из программирования. Но тогда как сама теорема, так и её доказательство теряют математическую строгость и становятся излишне лаконичными. Настолько, что восстановить точно доказательство становится затруднительным.

По этой причине формулировка основных понятий, относящихся к теореме Успенского-Райса, требует определения вычислительной модели, допускающей точные формулировки и доказательства. Существует несколько примеров таких вычислительных моделей. Самые известные – это машина Тюринга [2], [8], частично-рекурсивные функции [3], машина с неограниченными регистрами [1], [4], алгорифмы Маркова [5], машина Поста [6].

Формально говоря, это означает, что существуют различные по семантическому смыслу теоремы Успенского-Райса. Однако можно показать, что все эти вычислительные модели в известном смысле эквивалентны [6]. Несмотря на различные исходные понятия и различные системы определений сами доказательства теоремы для различных моделей по существу одинаковы.

Дадим предварительные определения, относящиеся к теореме Успенского-Райса. Уточнённые определения будут приведены ниже. Через N обозначим множество целых неотрицательных чисел (натуральных чисел). Пусть f – функция, область определения которой представляет собой подмножество N (или всё N), принимающая значение во множестве N. Функция f называется вычислимой, если существует программа p с одним входным параметром x из N, которая обладает следующими свойствами:

– если x принадлежит области определения функции f, то на входе x программа p выводит значение f(x) и останавливается;

– если x не принадлежит области определения функции f, то на входе x программа p зацикливается.

Про программу p мы будем говорить, что она вычисляет функцию f. Ясно, что существует бесконечное количество программ, вычисляющих одну и ту же функцию.

Под свойством программы мы будем понимать свойство функции, которую эта программа вычисляет. Отсюда следует, что две программы, вычисляющие одну и ту же функцию, обладают одинаковыми свойствами. Примером свойства будет следующая характеристика программы – функция, которую вычисляет программу, тождественно равна нулю. Пример характеристики, которая свойством не является, – программа содержит три команды.

Теорема Успенского-Райса (в очень приблизительной формулировке) утверждает, что наличие у программы определённого свойства нельзя определить по тексту программы. Точнее, имеется в виду следующее. Не существует программы, которая, получив на вход текст другой (исследуемой) программы, может определить: обладает ли исследуемая программа этим свойством или не обладает?

Доказательство теоремы Успенского-Райса,

как и сама теорема, имеется во многих источниках. Одним из основных источников по этой проблематике является [1].

Данные выше формулировки нуждаются в уточнении. Перечислим лишь некоторые из обстоятельств, требующих уточнения.

Если мы говорим о свойствах программ как о свойствах функций, которые программы вычисляют, то произвольная программа должна либо останавливаться, выводя при этом некоторое значение, или зацикливаться. Однако существуют программы, которые останавливаются и ничего не выводят, или которые останавливаются в результате возникновения исключительной ситуации. Такие программы никакую функцию не вычисляют.

Следующее обстоятельство состоит в том, что программы надо каким-то способом идентифицировать. Если фиксирован язык программирования, то программы можно идентифицировать, например, в лексикографическом порядке. Однако такая идентификация должна быть корректной в том смысле, что каждому номеру должна соответствовать программа. Для «обычного» языка программирования это не так – произвольная последовательность символов программой не является.

Поэтому необходима как формализация введённых понятий, так и формализованное изложение самих доказательств. Для настоящего изложения выбрана вычислительная модель, называемая машиной с неограниченными регистрами. Автор часто использовал конструкции из [4].

Раздел математики, к которому относится теорема Успенского-Райса, называется «теория алгоритмов». Книга [5] представляет собой фундаментальный обзор по теории алгоритмов. Профессиональный учебник по теории алгоритмов - это [3]. Начальные сведения можно получить в книге [7], ясно и доступно написанной.

2. Определение программы

Мы будем рассматривать вычислительную модель, которая называется «машина с неограниченными регистрами». Дадим основные определения. Машина с неограниченными регистрами содержит бесконечную последовательность регистров. В каждом регистре может храниться произвольное натуральное (целое неотрицательное) число.

Программой называется конечный последовательно занумерованный набор команд. Номер команды в программе будем называть *индексом* команды. Индексы команд программы начинаются с 1.

Выполнение программы состоит в том, что команды программы последовательно выполняются в порядке их индексации, начиная с первой команды. Порядок выполнения команд может быть изменён в результате выполнения команды условного перехода.

Выполнение команды, отличной от команды условного перехода, состоит в том, что команда изменяет значение (содержимое) одного из регистров R1, R2, … . Последовательность значений регистров называется конфигурацией.

Перед началом выполнения программы лишь конечное количество значений регистров отлично от нуля. Значение всех остальных регистров равно 0. Конфигурация, заданная перед началом выполнения программы, называется начальной.

Команды бывают четырёх типов: обнуления Z(n), прибавления единицы S(n), пересылки T(m,n) и условного перехода J(m,n,q). Опишем действие этих команд.

При выполнении команды Z(n) регистру Rn присваивается значение 0. Выполнение команды S(n) состоит в том, что значение регистра Rn увеличивается на 1. В результате выполнения команды T(m,n) значение регистра Rn становится равным значению регистра Rm. Значения всех остальных регистров, кроме регистра Rn, остаются прежними. Команда условного перехода J(m,n,q) предназначена для изменения порядка выполнения команд. Если значение регистра Rn равно значению регистра Rm, то следующей после команды J(m,n,q) будет выполняться команда с индексом q. Если значения регистров Rn и Rm различны, то после команды J(m,n,q) будет выполнена команда, находящаяся в программе вслед за командой J(m,n,q). Число q называется адресом перехода.

Числа m, n и q называются аргументами команды. Команды одного типа с различными аргументами считаются различными. Например, команды Z(1) и Z(2) различны.

Программа заканчивает работу (останавливается) в случае, если вслед за текущей должна быть выполнена команда, которая в программе отсутствует.

Одно из основных свойств машины с неограниченными регистрами состоит в том, что при любой начальной конфигурации программа либо останавливается, либо зацикливается. Если программа останавливается, то конфигурация, которая получилась после остановки, называется завершающей.

Пример. Программа, состоящая из одной команды J(1,1,1), всегда зацикливается. Программа Z(1) останавливается при произвольной начальной конфигурации.

Ниже под программой мы будем понимать

программу для машины с неограниченными регистрами. Также мы будем пользоваться термином «алгоритм». Под алгоритмом мы понимаем огрублённое описание последовательности вычислений, на основании которого можно написать программу.

3. Ввод и вывод

Аналогом ввода и вывода для программ машины с неограниченными регистрами является процедура чтения начальной конфигурации и формирования завершающей конфигурации. Особенность программ машины с неограниченными регистрами состоит в том, что программа использует столько регистров, сколько ей потребуется для вычислений.

Пусть $x_1, \ldots, x_n$ – последовательность из n натуральных чисел. Конфигурацию $x_1, \ldots, x_n, 0, 0, \ldots$ будем обозначать через $\langle x_1, \ldots, x_n \rangle$.

Если на начальной конфигурации $\langle x_1, \ldots, x_n \rangle$ программа p останавливается, то через $p(x_1, \ldots, x_n)$ будем обозначать значение регистра R1 после того, как программа p остановилась. В этом случае значение регистра R1 называется результатом выполнения (или выходом) программы p на начальной конфигурации (или на входе) $\langle x_1, \ldots, x_n \rangle$.

Пример. Программа

1 J(2,3,5)
2 S(1)
3 S(3)
4 J(1,1,1)

преобразует начальную конфигурацию <x,y> в завершающую конфигурацию <x+y,y,y>.

В качестве упражнения можно написать программы умножения чисел, вычитания, деления с остатком, возведения в целую степень.

Уточним определение вычислимой функции для машины с неограниченными регистрами. Функция f от n переменных называется *вычислимой*, если существует такая программа p, что выполнены следующие два условия:

1) для последовательности $x_1, \ldots, x_n$, для которой функция f определена, выполняется равенство $f(x_1, \ldots, x_n) = p(x_1, \ldots, x_n)$;

2) для последовательности $x_1, \ldots, x_n$, для которой функция f не определена, программа p на конфигурации $\langle x_1, \ldots, x_n \rangle$ зацикливается.

4. Композиция программ

Программным модулем (или просто модулем) называется последовательно индексированный набор команд. В отличие от программ, индексация команд в модуле может начинаться с произвольного положительного числа (не обязательно с единицы). Минимальный и максимальный индексы команд, образующих модуль, обозначаются через m_{min} и m_{max}. Ещё одно требование к модулю состоит в том, что адрес перехода q команды J(m,n,q) модуля должен удовлетворять неравенству $m_{min} \leq q$. Примером программного модуля является программа.

Программный модуль выполняет команды начиная с команды с индексом m_{min} (первой командой модуля). Программный модуль заканчивает выполнение, если следующей за текущей командой должна быть выполнена команда, индекс которой больше m_{max}.

При выполнении программный модуль начинает изменять конфигурацию, которая была при выполнении первой команды модуля. Эта конфигурация называется начальной конфигурацией модуля. Конфигурация, которая получается при окончании работы модуля, называется завершающей конфигурацией модуля.

Из двух модулей можно составить новый модуль. Для этого нужно, чтобы индекс последней команды предшествующего модуля был на единицу меньше индекса команды последующего модуля (тогда команды результирующего модуля будут последовательно индексированы).

Таким же способом, последовательно располагая модули друг за другом, из нескольких модулей можно составить один модуль. Процедура получения модуля из нескольких называется композицией.

Программный модуль называется нормализованным, если адреса перехода q всех команд J(m,n,q) модуля удовлетворяют условию $q \leq m_{max} + 1$.

Если программный модуль нормализованный, то после окончания выполнения модуля следующей будет выполнена первая команда последующего модуля.

Произвольный модуль может быть нормализован. Для этого команды J(m,n,q) модуля, у которых $q > m_{max} + 1$, надо заменить на команды $J(m,n, m_{max} + 1)$. Нормализация модуля не влияет на завершающую конфигурацию модуля.

Обычно программные модули получаются из программы сдвигом на несколько индексов «вниз», то есть индекс каждой команды увеличивается на одно и тоже для всех команд число $s \geq 1$. Для того, чтобы программный модуль, сдвинутый на s индексов, выполнялся так же, как и программа до сдвига, надо модифицировать все команды условного перехода J(m,n,q), заменив адрес перехода q на число q + s. В этом случае переходы будут происходить на те же команды, что и до сдвига.

Мы будем пользоваться следующими обозначениями. Если F – программа, то через F_{+s} будем обозначать программу, которая получается

из программы F сдвигом на s индексов вниз и заменой адресов перехода q на q + s.

5. Нумерация программ

Опишем формулы, устанавливающие взаимно однозначное соответствие между программами и натуральными числами.

Сначала определим номера команд. Номер команды с будем обозначать через #c. Для команд обнуления и прибавления единицы положим

$\#Z(n) = 4n$ и $\#S(n) = 4(n - 1) + 1$.

Из этих формул видно, что номера команд обнуления – это целые положительные числа, делящиеся на 4, а номера команд прибавления единицы – это целые положительные числа, которые при делении на 4 дают в остатке 1.

Для определения номеров команд пересылки и условного перехода определим функции N_2 и N_3. Положим

$N_2(k,l) = 2^{k-1}(2l - 1)$

и

$N_3(k,l,m) = N_2(k, N_2(l,m)) = 2^{k-1}(2^l(2m - 1) - 1)$.

Номера команд пересылки и условного перехода определяются формулами

$\#T(k,l) = 4(N_2(k,l) - 1) + 2$

и

$\#J(k,l,m) = 4(N_3(k,l,m) - 1) + 3$.

Номера этих команд представляют собой все целые положительные числа, которые при делении на 4 дают в остатке 2 и 3 соответственно.

Существует программа, которая по номеру команды определяет тип команды и её аргументы. Утверждение достаточно очевидное, однако мы поясним его на примере.

Тип команды будем обозначать целыми числами, лежащими в диапазоне от 0 до 3. Эти значения обозначают соответственно команды обнуления, прибавления единицы, пересылки и условного перехода.

Для определения типа команды, надо номер команды разделить на 4 с остатком. Затем, в зависимости от остатка, то есть от типа команды, надо вычислить аргументы команды. Например, если номер команды с равен 38, то остаток от деления #c на 4 будет 2. Следовательно, команда с номером 38 – это команда пересылки. Вычтя 2 из #c и разделив на 4, получим $N_2(k,l) = 2^{k-1}(2l - 1) = 10$. Разделив два раза $N_2(k,l)$ на 2 получим в остатке 1. Следовательно, k = 2. Поэтому 2l – 1 = 5 и l = 3.

Теперь определим номер программы. Пусть p – программа и $a_1, \ldots, a_n$ – последовательность номеров команд этой программы. Последовательность $b_1, \ldots, b_n$, элементы которой определяются формулой

$$b_k = a_1 + \ldots + a_k, \quad (1)$$

где k = 1, …, n, называется *последовательностью накопленных номеров* команд программы p. Так как номера команд – положительные числа, то последовательность накопленных сумм монотонно возрастает.

Из формулы (1) следует, что различным последовательностям номеров команд соответствуют различные последовательности накопленных номеров и что для каждой монотонно возрастающей последовательности положительных чисел существует единственная последовательность номеров команд, которая её порождает.

Номером #p программы p называется число

$$\#p = -1 + 2^{b_1-1} + \ldots + 2^{b_n-1}. \quad (2)$$

Поскольку последовательность накопленных номеров монотонно возрастающая, то представление номера программ в виде (2) однозначно.

Ясно также, что произвольное целое неотрицательное число представимо в виде (2). Поэтому целые неотрицательные числа находятся во взаимно однозначном соответствии программами машины с неограниченными регистрами.

Пример. Пусть программа p состоит из одной команды S(1). Тогда

$\#S(1) = a_1 = 1$, $b_1 = a_1 = 1$, $\#p = -1 + 2^{b_1-1} = 0$.

Опишем алгоритм программы, которая по номеру программы #p и индексу команды m в программе p определяет тип команды и её аргументы.

Для определения номера команды a_m надо последовательно определять накопленные номера $b_1, b_2, \ldots$. Накопленные номера вычисляются с помощью формулы (2) путём последовательного деления с остатком на 2 числа #p + 1. Определение очередного накопленного номера b_m может быть выполнено с использованием ограниченного, не зависящего от n, количества регистров. Поэтому это действие может быть выполнено с помощью программы машины с неограниченными регистрами.

Затем определяем номер команды $a_m = b_m - b_{m-1}$. Для определения номера a_m достаточно хранить текущий и предыдущий накопленные номера. Как было отмечено выше, существует программа, которая по номеру команды вычисляет её тип и аргументы. Поэтому, определив номер команды, можно вычислить эти данные.

Отметим, что наличие взаимно однозначного соответствия между натуральными числами и программами позволяет рассматривать программы машины с неограниченными регистрами как натуральные числа.

6. Универсальная функция

Определим функцию двух переменных Ф, аргументами которой являются целые неотрицательные числа. Через φ_x обозначим программу с номером x согласно определённой выше нумерации.

Если на входе $\langle y \rangle$ программа φ_x останавливается, то положим $\Phi(x,y) = \varphi_x(y)$. Если на входе $\langle y \rangle$ программа φ_x зацикливается, то функция Ф на паре аргументов (x,y) не определена. Функция Ф называется универсальной функцией (для вычислимых функций одного переменного).

Покажем, что универсальная функция вычислима. Вычислимость означает, что существует такая программа F, для которой на входе $\langle x, y \rangle$ выполняется равенство $\Phi(x,y) = F(x,y)$ в случае, если пара (x,y) принадлежит области определения функции Ф. Если же пара (x,y) не принадлежит области определения функции Ф, то программа F на входе $\langle x, y \rangle$ зацикливается.

Опишем алгоритм программы F. Для программы F нам потребуется целочисленное кодирование конфигураций. Мы используем кодирование отличное от кодирования, которое применялось для нумерации программ. Другой метод кодирования выбран потому, что действие команд машины с неограниченными регистрами на коды конфигураций для нового метода кодирования описывается проще, чем действие на коды, полученные прежним методом кодирования.

Определим этот метод кодирования. Обозначим через q_i простое число с номером i. Примеры: $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, и т.д. Согласно основной теореме арифметики произвольное целое положительное число $n \geq 2$ однозначно представимо в виде

$$n = p_1^{r_1} \times \ldots \times p_m^{r_m}, \qquad (3)$$

где r_i– натуральные числа и $r_m > 0$.

Пусть $r_1, \ldots, r_m$ – конечная последовательность натуральных чисел и $r_m > 0$. Поставим в соответствие конфигурации, порождённой последовательностью $r_1, \ldots, r_m$, число n согласно формуле (3). Отметим, что каждое такое n больше 1. Если конфигурация состоит из одних нулей, то положим n = 1. Число n называется номером конфигурации.

Опишем алгоритм, который по номеру конфигурации n и номеру k определяет число r_k. Первый шаг алгоритма состоит в определении простого числа q_k. Поскольку нас не интересует алгоритмическая сложность вычислений, то для определения простоты числа можно воспользоваться перебором и делением с остатком. Таким методом находим последовательно простые числа, пока не найдём простое число с номером k, то есть число q_k.

На следующем шаге алгоритма найдём r_k. Это можно сделать путём деления числа n на q_k с остатком. Если в результате деления с номером s остаток становится отличным от 0, то $r_k = s - 1$.

Теперь приведём алгоритм программы F, вычисляющий функцию Ф.

Вход для программы F – это конфигурация $\langle x, y \rangle$. В случае, когда пара (x,y) принадлежит области определения функции Ф, после завершения программы F на входе $\langle x, y \rangle$ значение регистра R1 должно быть равным $\varphi_x(y)$. В противном случае программа F на входе $\langle x, y \rangle$ зацикливается.

Программа F производит покомандный разбор φ_x и выполняет (интерпретирует) эти команды. При интерпретации каждой из команд φ_x программы F преобразуют не регистры R1, R2, …, которые используются самой программой F, а номер конфигурации, который хранится в переменной conf. В начале работы программы F переменная conf инициализируется значением 2^y, что равно номеру конфигурации $\langle y \rangle$.

Приведём псевдокод программы F. Пояснения помещаются в круглых скобках. В фигурных скобках указываются действия, которые последовательно должны быть совершены при выполнении определённого условия. Знак == используется для обозначения равенства двух величин. Знак := используется для обозначения того, что величине в левой части от этого знака присваивается значение, находящееся в правой части.

Начало алгоритма

Вход: $\langle x, y \rangle$

По номеру x определяем n (n – количество команд программы φ_x)

i := 1 (i – индекс текущей команды программы φ_x)

conf := 2^y (conf – номер текущей конфигурации)

(конфигурация инициализируется начальным значением y,0,0,…)

Начало цикла

Если i > n, то переходим на первую команду программы после окончания цикла

Определяем тип и аргументы команды с индексом i программы φ_x

(тип команды записывается в переменную type)

(для команд Z,S,T и J переменная type принимает значения 0,1,2,3)

(аргументы команд записываются в переменные m, n и q)

Если type == 0, то

{Находим p_m и r_m в разложении conf на простые сомножители

conf := conf/($p_m^{r_m}$)

i := i + 1

Переходим на начало цикла}

Если type == 1, то

{Находим p_m в разложении conf на простые сомножители

conf := conf× p_m

i := i + 1

Переходим на начало цикла}

Если type == 2, то

{Находим r_m в разложении conf на простые сомножители

Находим p_n и r_n в разложении conf на простые сомножители

conf := $(conf/(p_n^{r_n}))\times(p_n^{r_m})$}

i := i + 1

Переходим на начало цикла}

Если type == 3, то

Находим r_m в разложении conf на простые сомножители

Находим r_n в разложении conf на простые сомножители

i := i + 1

Если $r_m == r_n$, то i := q

Переходим на начало цикла}

Конец цикла

(Попадаем сюда, если вышли из цикла)

Находим r_1 в разложении conf на простые сомножители.

В R1 записываем r_1

Конец алгоритма

Итак, мы описали алгоритм программы F, которая вычисляет универсальную функцию Φ. В силу того, что алгоритм использует только конечное количество регистров, которое не зависит от начальной конфигурации $\langle x, y\rangle$, алгоритм может быть реализован в виде программы на языке машины с неограниченными регистрами.

7. Диагональная конструкция

Построим пример функции, которая не является вычислимой. Пусть Φ определённая выше универсальная функция и x – натуральное число. В случае, если пара (x,x) принадлежит области определения функции Φ, положим $\zeta(x) = \Phi(x,x) + 1$. Если пара (x,x) не принадлежит области определения функции Φ, то положим $\zeta(x) = 0$.

Функция ζ не является вычислимой. Предположим, что это не так. Тогда существует такое натуральное n, что программа φ_n вычисляет эту функцию. Если на входе n программа φ_n останавливается, то должно выполняться равенство

$\varphi_n(n) = \Phi(n,n) + 1$ или $\varphi_n(n) = \varphi_n(n) + 1$,

что невозможно. Если же на входе n программа φ_n зацикливается, то равенство $\varphi_n(n) = \zeta(n)$ также невозможно, поскольку левая часть этого равенства не определена, а правая равна 0. Итак, мы доказали, что функция ζ не является вычислимой.

Теперь определим вычислимую функцию d. Для тех x, для которых пара (x,x) принадлежит области определения функции Φ, положим $d(n) = \Phi(n,n)$. Если же пара (x,x) не принадлежит области определения функции Φ, то функция d для аргумента x не определена. Так определённую функцию d будем называть диагональной.

Покажем, что функция d вычислима. Пусть F – программа, вычисляющая функцию Φ. Через F_{+1} обозначим модуль, который получается из программы F сдвигом на 1. Программа G, состоящая из команды T(1,2) и модуля $F_{+1,}$ вычисляет функцию d.

Начальная конфигурация для программы G – это $\langle x\rangle$. Команда T(1,2) превращает конфигурацию $\langle x\rangle$ в конфигурацию $\langle x, x\rangle$. Модуль F_{+1} вычисляет функцию d на паре аргументов (x,x), то есть выводит число Φ(x,x), если x принадлежит области определения функции d, и зацикливается в другом случае. Следовательно, программа F вычисляет функцию d.

Диагональная функция не является всюду определённой. Это следует, например, из того, что программа, состоящая из одной команды J(1,1,1), зацикливается.

8. Разрешимые множества

Пусть S – некоторое множество натуральных чисел N. Характеристической функцией множества S называется всюду определённая функция, которая на элементах множества S принимает значение 1, а на элементах из N, не принадлежащих S, принимает значение 0. Множество S называется разрешимым, если характеристическая функция этого множества вычислима.

Покажем, что область определения D диагональной функции d не является разрешимым множеством.

Предположим, что характеристическая функция χ множества D вычислима и что G – программа, вычисляющая функцию χ. Без ограничения общности можно считать, что программа G не изменяет и не использует в вычислениях значение регистра R2. Кроме того, программа G при завершении обнуляет все регистры, кроме регистров R1 и R2. Результат выполнения программы G содержится в R1.

Обозначим через G_{+1} модуль, который получается из программы G сдвигом на единицу. Через F обозначим программу, которая вычисляет функцию Φ.

Покажем, что из предположения вычислимости функции χ следует существование программы, вычисляющей функцию ζ. Приведём алгоритм этой программы.

Начало алгоритма

Вход: $\langle x\rangle$

T(1,2) (запоминаем x в регистре R2)

G_{+1} (в R1 записывается 0 или 1)

J(1,3,q) (значение R3 равно 0, значение q больше индекса команды S(1))

T(2,1) (конфигурация становится равной $\langle x, x\rangle$)

F_{+s} (s такое, что модуль F_{+s} находится сразу за командой T(2,1))

S(1) (значение регистра R1 становится равным Φ(x,x) + 1)

Конец алгоритма

Итак, в предположении, что множество D разрешимо, мы написали программу, вычисляющую ζ. Так как функция ζ не является вычислимой, то предположение о том, что множество D разрешимо неверно.

Лемма. Пусть f – вычислимая функция двух переменных. Тогда существует такая всюду определённая вычислимая функция одного переменного к, что $f(x,y) = \varphi_{к(x)}(y)$.

Точнее, если пара (x,y) принадлежит области определения функции f, то выполнено приведённое выше равенство. Если же пара (x,y) не принадлежит области определения функции f, то программа $\varphi_{к(x)}$ на входе $\langle y\rangle$ зацикливается.

Доказательство. Сначала дадим набросок доказательства. Пусть F – программа, которая вычисляет функцию f. Если в программе F зафиксировать значение первой переменной, то получится программа, зависящая только от одного входного параметра. Каждая такая программа имеет свой номер, который зависит от значения фиксированной переменной. Эта зависимость и представляет собой искомую функцию к.

Опишем процедуру построения функции к подробнее. Для каждого натурального a построим программу X_a, которая обладает следующим свойством: если на паре аргументов (a,y) функция f определена, то программа X_a на конфигурации $\langle y\rangle$ выводит в R1 значение функции f(a,y); если же на этой паре аргументов функция f не определена, то программа X_a зацикливается. Приведём алгоритм программы X_a.

Начало алгоритма

T(1,2) (конфигурация $\langle y\rangle$ превращается в $\langle y, y\rangle$)

Z(1) (конфигурация превращается в $\langle 0, y\rangle$)

S(1) (эта команда выполняется a раз, конфигурация становится равной $\langle a, y\rangle$)

F_{a+2} (модуль вычисляет функцию f(a,y))

Конец алгоритма

Результат выполнения модуля F_{a+2} на конфигурации $\langle a, y\rangle$ совпадает с результатом выполнения программы F на той же конфигурации. Поэтому если функция f определена на паре аргументов (a,y), то результат выполнения модуля F_{a+2}, а значит и программы X_a будет равен f(a,y). В противном случае программа F на этой конфигурации зацикливается.

Определим номер программы X_a. Номера первых a + 2 команд программы X_a составляют последовательность 10, 4, 1, …, 1. Количество чисел 1 в этой последовательности равно a. Далее следуют команды модуля F_{a+2}.

Обозначим номера команд программы F через a_1, …, a_n. Команды программы F с индексом i, отличные от команд перехода, совпадают с командами модуля F_{a+2} с индексом i + a + 2. Поэтому номера этих команд в программе F и в модуле F_{a+2} совпадают.

Найдём разность номеров для команд перехода модуля F_{a+2} и программы F. Непосредственное вычисление показывает, что

$\#J(m,n,k+q) - \#J(m,n,q) = 2^{m+n+2}q$.

В нашем случае q = a + 2. Поэтому номера команд модуля F_{a+2}, начиная с команды с номером a + 3, образуют последовательность

$d_1 = a_1 + c_1 q$, …, $d_n = a_n + c_n q$.

Для команд Z(n), S(n) и T(m,n) коэффициенты c_i равны 0, а для команд J(m,n,k) коэффициенты c_i равны $2^{m+n+2}q$.

В соответствии с этими формулами можно написать программу, которая последовательно вычисляет номера команд программы X_a. Параллельно с вычислением номеров программы X_a можно, согласно формулам (1), последовательно вычислять накопленные номера b_1, …, b_{n+a+2} команд программы X_a, и частичные значения суммы для определения номера программы по формуле (2). Полученное в результате этих вычислений значение и будет номером к(a) программы X_a.

Итак, мы показали, что значение к(a) вычислимо зависит от a. Согласно определению программа X_a совпадает с программой $\varphi_{к(a)}$. Следовательно, $\varphi_{к(a)}$ вычисляет функцию f(a,y). *Лемма доказана*.

9. Доказательство теоремы Успенского-Райса

Вначале приведём точную формулировку теоремы Успенского-Райса. Мы рассматриваем программы, входом которых является натуральное число. Программы p_1 и p_2 назовём эквивалентными, если они вычисляют одну и ту же функцию. Точнее, для любого общего входа $\langle x\rangle$ программы p_1 и p_2 останавливаются (каждая из них), либо обе программы зацикливаются. Если программы останавливаются, то выполняется равенство $p_1(x) = p_2(x)$. Все программы разбиваются на классы эквивалентности согласно введённому определению эквивалентности.

Свойством назовём множество классов эквивалентности программ. Будем говорить, что программа p обладает свойством A, если класс эквивалентности программы p принадлежит A. Множество номеров программ, которые обладают свойством A, будем обозначать через S(A).

Свойство A называется нетривиальным, если оно не совпадает с множеством всех классов эквивалентности и не пусто.

Пример свойства: класс эквивалентности состоит из программ, которые зацикливаются на произвольном входе.

Свойство A называется разрешимым, если множество номеров программ S(A), которые этим свойством обладают, разрешимо.

Теорема (*Успенского-Райса*). Произвольное нетривиальное свойство программ неразрешимо.

Доказательство. Пусть A – нетривиальное свойство. Через B обозначим дополнительное свойство, состоящее из классов эквивалентности, которые не принадлежат A.

Будем предполагать, что программа, которая зацикливается на любом входе, обладает свойством A. Пусть Q – программа, обладающая свойством B. Программа Q существует, поскольку свойство A нетривиально. Через q обозначим функцию, которую вычисляет программа Q. В случае, когда Q останавливается на конфигурации $\langle y \rangle$, должно выполняться равенство q(y) = Q(y). Если программа Q зацикливается на конфигурации $\langle y \rangle$, то функция q при значении аргумента y не определена.

Обозначим через D область определения диагональной функции d. Напомним, что D не совпадает с множеством N и что D не является разрешимым множеством. Программу, вычисляющую функцию d, обозначим через P.

Определим функцию f двух аргументов x и y. Если x принадлежит D, и y принадлежит области определения функции q, то положим f(x,y) = q(y). В остальных случаях функция f на паре аргументов (x,y) не определена.

Покажем, что функция f вычислима. Программа F, которая вычисляет функцию f, представляет собой композицию программы P и модуля $\overline{Q}$. Модуль $\overline{Q}$ состоит из команды T(2,1) и модуля Q_{+s}. Сдвиг s выбран так, чтобы первая команда модуля Q_{+s} имела индекс на единицу больше, чем команда T(2,1). Без ограничения общности можно предполагать, что программа P не изменяет и не использует значение регистра R2.

Программа F вычисляет f. Действительно, пусть $\langle x, y \rangle$ – начальная конфигурация. Если x принадлежит D, то программа P завершает выполнение и начинает выполняться модуль $\overline{Q}$. Этот модуль сначала переписывает в R1 значение y, а затем запускается модуль Q_{+s}, который вычисляет функцию q. Если x не принадлежит D, то программа P на этом входе зацикливается.

Поскольку f – вычислимая функция двух аргументов, то согласно лемме существует такая вычислимая функция к, что для каждого y, для которого функция q определена, выполняется равенство $q(y) = \varphi_{к(x)}(y)$. Если q не определена на y, то программа $\varphi_{к(x)}$ на входе $\langle y \rangle$, зацикливается.

Это означает, что при x из D программа с номером к(x) эквивалентна программе Q. Поэтому к(x) принадлежит S(B). Если же x не принадлежит D, то программа $\varphi_{к(x)}$ зацикливается на произвольном входе. Следовательно, эта программа обладает свойством A, и к(x) принадлежит S(A).

Предположим, что множество S(A) разрешимо. Тогда существует вычислимая функция χ, принимающая на S(A) значение 1, а на S(B) значение 0.

Функция χ∘к будет всюду определённой. Также эта функция будет вычислимой. Действительно, пусть программа K вычисляет функцию к, а программа X вычисляет функцию χ. (Здесь через K мы обозначили прописную букву каппа, а через X - прописную букву хи).

Обозначим через X_{+s} модуль, который получается из программы X сдвигом, чтобы команды модуля X_{+s} начинались сразу за командами программы K. Тогда композиция программы K и модуля X_{+s} будет вычислять функцию χ∘к. Здесь мы делаем обычное предположение о нормализованности программы K.

По построению функция χ∘к будет характеристической вычислимой функцией множества D, что невозможно, поскольку множество D неразрешимо. Следовательно, предположение о том, что множество S(A) разрешимо, неверно. *Теорема доказана*.

10. Заключение

Автор благодарит С.Ф. Сопрунова за тщательное чтение начального варианта статьи и ценные замечания, существенно улучшившие изложение, Ю.В Кузнецова и М.В. Михайлюка за внимание к этой публикации и благожелательные и конструктивные комментарии.

On Uspensky-Rice Theorem

Andrey Gryuntal

Abstract. This paper contains detailed proof of the Uspensky-Rice theorem. Basic definitions related to computable functions are introduced. The computing model of an unlimited register machine is used. The presentation is of thorough and elementary nature.

Keywords: computable functions, program, unlimited register machine

Литература

1. Н.К. Верещагин, А. Шень, Вычислимые функции, М., МЦНМО, 2017.
2. С. Пилипенко, Дискретная математика 2. Конспект, 2020-2021, https://hse-tex.me/course-2/discrete-math-02-lecture-notes.pdf.
3. В.А. Успенский, Лекции о вычислимых функциях, М., Государственное издательство физико-математической литературы, 1960.
4. Ю.Ю. Кочетков, Вычислимые функции, http://kirill-andreyev.narod2.ru.
5. В.А. Успенский, А.Л. Семёнов, Теория алгоритмов: основные открытия и приложения, М., «Наука», Главная редакция физико-математической литературы, 1987.
6. А.Е. Ромащенко, https://users.mccme.ru/anromash/courses/lecture-notes-logic-computability-2013.pdf
7. В.А. Успенский, Машина Поста, Популярные лекции по математике, выпуск 54, издание второе, М., «Наука», главная редакция физико-математической литературы, 1988.
8. Конспект лекций О.Б. Лупанова по курсу «Введение в математическую логику», М., МГУ им. М.В. Ломоносова, механико-математический факультет, 2007.

Подписано в печать 28.11.2024 г.
Формат 60x90/8
Печать цифровая. Печатных листов 13
Тираж 100 экз. Заказ № 1528

Отпечатано в ФГБУ «Издательство «Наука»
(Типография «Наука»)
121099, Москва, Шубинский пер., 6