МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий
Кафедра параллельных вычислений

Направление подготовки 09.03.01 Информатика и вычислительная техника
Направленность (профиль): Программная инженерия и компьютерные науки

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Морозова Руслана Сергеевича

Тема работы:

РАЗРАБОТКА И РЕАЛИЗАЦИЯ АЛГОРИТМА ПЛАНИРОВАНИЯ НА ВЫЧИСЛИТЕЛЬНЫХ МОДЕЛЯХ С МАССИВАМИ

«К защите допущена»
Заведующий кафедрой,
д.т.н., профессор
Малышкин В.Э. /……..
(ФИО) / (подпись)
« » _____ 2024 г.

Руководитель ВКР
д.т.н., профессор,
зав. каф. ПВ ФИТ НГУ
Малышкин В.Э. /……...
(ФИО) / (подпись)
« » _____ 2024 г.

Соруководитель ВКР
к.т.н.,
ст. преп. каф. ПВ ФИТ НГУ
Перепёлкин В.А. /……..
(ФИО) / (подпись)
« » _____ 2024 г.

Новосибирск, 2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)
Факультет информационных технологий
Кафедра параллельных вычислений

Направление подготовки 09.03.01 Информатика и вычислительная техника
Направленность (профиль): Программная инженерия и компьютерные науки

УТВЕРЖДАЮ

Зав. кафедрой Малышкин В.Э.
(фамилия, И., О.)
…….
(подпись)
«03» ноября 2023 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

Студенту Морозову Руслану Сергеевичу, группы 20203
(фамилия, имя, отчество, номер группы)

Тема «Разработка и реализация алгоритма планирования на вычислительных моделях с массивами»
(полное название темы выпускной квалификационной работы)

утверждена распоряжением проректора по учебной работе от 03.11.2023 № 0413

Срок сдачи студентом готовой работы 20 мая 2024 г.

Исходные данные (или цель работы): реализовать систему планирования на вычислительных моделях с массивами, использующую разработанные алгоритмы планирования и способы описания задач на вычислительных моделях.

Структурные части работы: анализ предметной области, описание алгоритма планирования, программная реализация системы планирования, выбор практически значимых задач, тестирование системы.

Руководитель ВКР
зав. каф. ПВ ФИТ НГУ,
д.т.н, профессор
Малышкин В.Э. /……
(ФИО) / (подпись)
«03» ноября 2023 г.

Задание принял к исполнению
Морозов Р.С. /……
(ФИО студента) / (подпись)
«03» ноября 2023 г.

Соруководитель ВКР
ст. преп. каф. ПВ ФИТ НГУ
к.т.н.
Перепёлкин В.А. /……
(ФИО) / (подпись)
«03» ноября 2023 г.

СОДЕРЖАНИЕ

Определения, обозначения и сокращения ... 5
Введение ... 6
1 Обзор предметной области ... 9
1.1 Простые вычислительные модели ... 9
1.2 Переход к массовому случаю ... 10
1.3 Другие результаты ... 10
1.4 Итог обзора ... 11
2 Алгоритм планирования ... 13
2.1 Постановка задачи ... 13
2.1.1 Необходимые определения ... 13
2.1.2 Требования к решению ... 15
2.2 Идея решения ... 16
2.3 Планирование в несколько этапов ... 17
2.4 Способ представления счётных множеств ... 18
2.4.1 Описание подхода к представлению ... 18
2.4.2 Описание расширения для многомерного случая ... 20
2.5 Описание алгоритма планирования ... 21
2.5.1 Общее представление входных данных ... 22
2.5.2 Планирование на общей задаче ... 22
2.5.2.1 Восходящая часть планирования ... 23
2.5.2.2 Алгоритм имитации планирования ... 24
2.5.2.3 Алгоритм определения применимости операции к множеству переменных ... 26
2.5.2.4 Алгоритм попытки построения пси-операции ... 26
2.5.2.5 Алгоритм построения идентичных термов ... 28
2.5.2.6 Нисходящая часть планирования ... 29
2.5.3 Планирование на частной задаче ... 30
2.5.3.1 Нисходящая часть планирования ... 30
2.5.3.2 Восходящая часть планирования ... 31
2.6 Анализ решения ... 31
2.6.1 Ключевые особенности решения ... 31
2.6.2 Оценка корректности ... 32
2.6.3 Оценка алгоритмической сложности ... 33
2.6.4 Недостатки ... 33
3 Программная реализация системы планирования ... 35
3.1 Описание библиотеки планирования ... 35
3.1.1 Архитектура ... 35
3.1.2 Ключевые классы ... 36
3.1.3 Особенности реализации ... 39
3.2 Описание сервиса планирования ... 39
3.2.1 Использование в системе LuNA ... 39

3.2.2 JSON формат вычислительных моделей с массивами 40
3.3 Тестирование 42
3.3.1 Подход к тестированию 42
3.3.2 Задача сейсмической свёртки 42
3.3.3 Задача моделирования потока жидкости 46
3.3.4 Результаты тестирования 50
Заключение 52
Список использованных источников и литературы 54
Приложение А Алгоритм сокращения множества элементов массива 57
Приложение Б Описание программы 62
Приложение В Руководство программиста 73
Приложение Г Задача вычисисления чисел Фибоначчи 79

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ВМ	Вычислительная модель
ВММ	Вычислительная модель с массивами

ВВЕДЕНИЕ

Многие задачи численного моделирования требуют высоких вычислительных мощностей. Часто существенное ускорение может быть получено при использовании параллельных программ [1]. Непосредственно создание параллельных программ с помощью стандартных средств [2,3] требует от программиста специальных знаний [4]. Учёным и исследователям, использующим численное моделирование как инструмент, сложно создавать параллельные программы вручную, поэтому им нужны системы, автоматически конструирующие параллельные программы.

Один из используемых подходов – синтез параллельных программ на вычислительных моделях [5]. Вычислительные модели, позволяют описать часть предметной области в виде переменных и операций над ними. Таким образом появляется возможность создания систем активных знаний [6], удобных для решения вычислительных задач в науке и промышленности. В отличие от моделей вычислений с множественным присваиванием, вычислительные модели задают отношения между переменными, что позволяет автоматически определять зависимости по данным для конструирования параллельных программ.

В общем случае вычислительная модель описывает больше объектов, чем нужно для решения конкретных задач, поэтому необходимо выполнять планирование – выделять подмножество операций, достаточное для вычисления требуемых переменных.

На практике часто встречаются предметные области с множествами однотипных объектов, которые удобно представлять в виде массивов. Поэтому отдельно в [5] вводятся вычислительные модели с массивами, которые, в частности, описывают счётные множества объектов и операций предметной области.

С использованием массовых операций планирование становится сложнее. Алгоритм, описанный в [7], предлагает способ планирования для таких случаев.

Поскольку алгоритм описан в понятиях теории множеств, можно столкнуться с рядом сложностей при его программной реализации.

Целью работы является разработка и реализация алгоритма планирования на вычислительных моделях с массивами. Алгоритм должен удовлетворять следующим требованиям:

а) Должна быть возможность описать множества однотипных объектов в виде одномерных массивов, и определить массовые операции над ними.

б) Массовые операции должны поддерживать индексацию используемых массивов с любыми линейными функциями от индекса.

в) Алгоритм должен быть применим для некоторых классов практически значимых задач, то есть должна быть возможность ограничить вычисления.

г) Сложность алгоритма должна зависеть от сложности описываемых отношений в ВМ, а не от количества объектов.

Для достижения цели необходимо выполнить **задачи**:

а) Выполнить анализ существующих подходов и решений.

б) Выделить классы практически значимых задач, представимых в виде ВММ.

в) Разработать детализированное описание алгоритма планирования на ВММ.

г) Реализовать систему планирования и форматы описания входных и выходных данных.

д) Выполнить тестирование системы.

Научная новизна работы заключается в детализации существующих алгоритмов планирования на ВММ с разработкой способа представления счётных множеств в памяти ЭВМ, а также в разработке упрощённого формата описания ВММ, практически подкреплённого алгоритмом планирования.

Практическая ценность работы состоит в том, что разрабатываемая система может участвовать в автоматическом конструировании параллельных для актуальных задач численного моделирования по относительно простому их описанию.

Работа состоит из введения, трёх глав и заключения. В первой главе даётся определение предметной области, делается критический обзор существующих решений. Во второй главе даётся постановка задачи, теоретические описание и анализ предлагаемого решения. Третья глава посвящена описанию реализации решения и практических результатов, достигнутых с помощью этого решения. В заключении сформулированы основные выводы работы и дальнейшие планы.

1 Обзор предметной области

В разделе проводится анализ существующих подходов к описанию ВМ и планированию на них. Решения рассматриваются с точки зрения выразительности используемых средств описания ВМ, а также возможности использования идей алгоритмов для планирования на ВММ.

Также рассматриваются готовые системы, участвующие в разных этапах процесса синтеза параллельных программ на вычислительных моделях. Оценивается возможность их применения для планирования на ВММ.

1.1 Простые вычислительные модели

Большинство исследований в этой области основываются на работе [8]. Тыугу предложил способ описания задач в виде вычислительных моделей. Для задания отношений между переменными он использовал математические уравнения. В частности, был описан пример использования для решения простой геометрической задачи. В этой работе же он и предложил базовый алгоритм планирования вычислений, который определяет какие операции нужно выполнить для решения задачи. Причём рассмотрен вариант планирования для нестационарных вычислительных моделей, где применимость отношений зависит от значений переменных.

Данная идея была далее раскрыта в системе ПРИЗ [9]. В этой работе предложен язык формального описания задач на вычислительных моделях. Планирующими компонентами системы являются генератор программ и синтезатор.

Ещё одна работа [10] рассматривает способ планирования для класса задач с однородными отношениями порядка один. Предлагается планирование, основанное на теоремах из теории графов, которое позволяет находить решение за линейную сложность ($O(k)$, где k-количество операций). Однако недостаток такого подхода в том, что он существенно сужает класс решаемых задач, и не подходит для массового случая.

В работе [11] предлагается алгоритм, выполняющий планирование на основе моделирования исполнения, учитывающий нефункциональные свойства программных модулей и системы-исполнителя. Если ВММ преобразовывать в эквивалентную простую ВМ, можно использовать данный подход для синтеза эффективных параллельных программ.

1.2 Переход к массовому случаю

В работе [5] вводится понятие упрощённой вычислительной модели с массивами. Там рассматриваются абстрактные задачи, которые демонстрируют выразительность такого формата. Сам алгоритм планирования подробно описан в отдельной работе [7]. Предлагаемый подход использует одномерные массивы и линейную индексацию массивов в операциях, что существенно ограничивает класс решаемых задач. Также отсутствует возможность ограничить вычисления на этапе планирования, поскольку алгоритм работает только со счётными последовательностями. Важно отметить, что доступные реализации этого алгоритма отсутствуют, а непосредственно реализация имеет ряд трудностей, связанных с представлением счётных множеств в памяти ЭВМ. Из-за этого трудно оценить, какие классы задач могут решаться предложенным способом.

На базе подхода разработана система фрагментированного программирования LuNA [12], которая использует при описании программ вычислительные модели. Однако задача планирования в системе не решается, поскольку программа в системе обязана задавать уже спланированную задачу на вычислительной модели. При этом система может параллельно исполнять независимые операции, в том числе и описанные в массовым виде, что может быть использовано для исполнения параллельных программ на основе ВММ.

1.3 Другие результаты

В работе [5] также рассматривается и общий вариант вычислительных моделей с массивами, в котором присутствуют предикаты и счётчики. Однако алгоритм планирования, который может быть применён в таком случае, не

описывается. Поэтому, несмотря на высокую выразительность описания ВМ, трудно оценить применимость подхода к решению реальных задач.

Близкий к ВМ подход основан на теории структурных функциональных моделей [13]. В нём возможность описания счётных множеств вычислений реализуется за счёт наличия рекурсивных выражений [14]. Также есть поддержка предикатов. При помощи интерпретации связи преобразуются в операции и условия ветвления, образуя логическую программу. Существует возможность определить для такого результата точки распараллеливания [15]. Однако такой формат не может в полной мере покрыть цели ВММ, поскольку все объекты и условия должны быть обработаны явно для вывода вычислимости.

В проекте HPC Community Cloud одна из подсистем – Aka – является онлайн-IDE для разработки вычислительных моделей и решения задач на основе вычислительных моделей [16]. В системе используются простые вычислительные модели, а также рекурсивные вычислительные модели. Данный проект демонстрирует большое количество разработок в областях хранения ВМ и задач, запуска и мониторинга процесса вычислений. Также используется способ задания ограничений вычисления во время исполнения – часть выходных переменных может быть не вычислена операцией, и дальнейшие вычисления будут перепланированы [17]. Возможности системы можно расширять для использования ВММ, либо использовать наработки проекта при организации работы с ВММ.

1.4 Итог обзора

Исходя из проведённого анализа, можно сделать вывод, что существует множество подходов к описанию ВМ, покрывающих разные классы задач. Наиболее выразительными являются вычислительные модели с массивами. Однако на практике даже для упрощённых ВММ не используются уже описанные алгоритмы планирования. Непосредственно реализация и экспериментальное исследование таких алгоритмов может помочь в дальнейшем развитии области и расширении класса описываемых рассматриваемыми

подходами задач. Важным практическим результатом будет разработка более примитивных и простых для программной реализации способов ограничения вычислений на простых ВММ.

Другой класс ВМ использует предикаты или другие аналоги условий ветвления. Однако не описаны алгоритмы для планирования в массовых случаях. Поэтому необходимо сначала использовать на практике простые ВММ и определить, как можно совместить эти подходы в дальнейшем.

Также существует ряд систем, которые могут быть объединены для получения единой системы исполнения параллельных программ по массовому описанию объектов в различных областях численного моделирования.

2 Алгоритм планирования

2.1 Постановка задачи

2.1.1 Необходимые определения

Множества и функции, входящие в определение вычислительных моделей с массивами [5]:

Множество переменных **X** – объединение конечного множества простых переменных и элементов конечного числа счётных линейно упорядоченных множеств (*массивов*). Простые переменные и массивы именованы, а элементы массивов также имеют порядковый индекс в пределах данного множества (массива) и называются *компонентами*.

Множество операций **F** – подмножество декартового произведения конечного множества функциональных символов (операций) и множества натуральных чисел. При этом, если операция присутствует только в одном экземпляре, она является простой и определяется только своим именем. В противном случае операция задаёт массовую операцию, и индекс задаёт *вхождение* этой операции.

Две функции **In** и **Out** сопоставляют каждому вхождению (или простой операции) конечные подмножества **X**, называемые множествами *входных* и *выходных* переменных соответственно. Любой элемент **X**, входящий в множества входных или выходных переменных вхождения, называется *используемым* во вхождении. Множество используемых переменных задаёт *сигнатуру* операции. При этом накладываются следующие ограничения:

– Сигнатуры вхождений одной операции совпадают;

– Разные вхождения одной операции используют разные компоненты одних и тех же массивов (либо простые переменные), причём зависимость используемого компонента от вхождения можно описать линейной функцией от номера вхождения.

Будем говорить, что операция *f вычисляет* переменную *x*, если *f* использует *x* как выходную переменную, либо существует другая операция, вычисляющая *x*

и использующая как входную одну из выходных переменных f. Все входные переменные вычисляющих x операций также будем считать вычисляющими её.

Итак, вычислительная модель с массивами – набор (**X**, **F**, **In**, **Out**).

Задача на вычислительной модели – это набор из двух конечных подмножеств **X** – **V** и **W** – для входных и выходных переменных задачи соответственно.

Также необходимо ввести множество *термов* T_0:

а) Если $x \in X$, то $t \coloneqq x \in T_0$, $\mathrm{in}(t) = \mathrm{out}(t) = \{\mathrm{x}\}$, $\mathrm{var}(\mathrm{t}) = \{\mathrm{x}\}$, $\mathrm{top}(\mathrm{t}) = \mathrm{x}$. Будем обозначать подобные термы как t_x.

б) Если $a_j \in F$, $\mathrm{in}(a_j) = \{x_1, x_2, \dots, x_n\}$, $t_1, \dots, t_n$ – такие термы из T_0, что $x_i \in out(t_i)$, $i = 1,2, \dots, n$, то $\mathrm{t} \coloneqq a_j(t_1, t_2, \dots, t_n) \in T_0$, $out(t) = out(a_j)$, $\mathrm{in}(t) = \bigcup_{i=1}^{\mathrm{n}} in(t_i)$, $\mathrm{var}(\mathrm{t}) = \bigcup_{\mathrm{i}=1}^{\mathrm{n}} \left(\mathrm{var}(\mathrm{t_i}) \setminus out(\mathrm{t_i})\right) \cup \left(in(a_j) \cup out(a_j)\right)$, $top(t) = a_j$. Будем обозначать подобные термы как t_a.

Множество *подтермов* $st(t)$ терма t определяется как:

а) $st(t_x) = \{x\}$;

б) $\mathrm{st}(t_a) = \{\mathrm{t}\} \cup (\bigcup_{i=1}^{\mathrm{n}} st(t_i))$.

Глубина $d(t)$ терма t определяется как:

а) $d(t_x) = 0$;

б) $d(t_a) = \max_i\left(d(t_i)\right) + 1$.

В алгоритме планирования используется обход в ширину и новые вхождения могут достигаться на разных итерациях обхода. Если на i-ой итерации встретилось n-ое вхождение операции a, то обозначим это как a_i^n. В таком случае *идентичность* термов t_1 и t_2 определяется следующим образом:

а) Если $t_1 = a_i^n(x_1, \dots, x_m)$, $t_2 = a_j^k(y_1, \dots, y_m)$, $n \neq k$, то t_1 и t_2 идентичны.

б) Если $t_1 = a_i^n(t_1', \dots, t_m')$, $t_2 = a_j^k(t_1'', \dots, t_m'')$, термы t_i' и t_i'' попарно либо идентичны, либо глубины 0 для $i = 1,2, \dots, m$, то t_1 и t_2 идентичны тогда и только

тогда, когда при $i \geq j$ ($i \leq j$) уровень вхождения $\mathrm{top}(t'_n)$ не меньше (не больше) уровня вхождения $\mathrm{top}(t''_n)$ для любого $n = 1,2,\dots,m$.

Терм называется *бесповторным*, если на любом пути от листа к корню каждое вхождение операции встречается не более одного раза для вычисления каждой своей выходной переменной. Древовидная структура связана с рекурсивным определением терма.

Счётное множество термов $\{t_1, t_2, \dots\}$ задаётся *параметрическим термом* $t(i)$, если выполнено одно из условий:

а) если $t(i) = t_x(i) = x_{f(i)}$, то $t_i = t(i)$ для $i \in \mathbb{N}$;

б) если $\mathrm{t(i)} = \mathrm{t_a(i)} = \mathrm{a}_{f(\mathrm{i})}\left(\mathrm{t}'_1(\mathrm{i}), \dots, \mathrm{t}'_m(\mathrm{i})\right)$, $t'_j(i)$ $j = 1,2 \dots, m$ – другие параметрические термы, то $t_i = t(i)$ для $i \in \mathbb{N}$.

Параметрический терм называется *самоподпитывающимся* относительно множества переменных V, если для любого $\mathrm{k} = 1,2,\dots$

$$in(t_k) \subseteq \bigcup_{m=1}^{k-1} var(t_m) \cup V.$$

2.1.2 Требования к решению

Основная задача алгоритма планирования – получить по задаче на ВМ подмножество **F** – план, – которое задаёт вычисление переменных из **W** по заданному множеству **V**. В данной работе рассматриваются ВМ, не зависящие от означивания переменных, поэтому план может быть вычислен сразу же при определении задачи на ВМ.

Одной из задач, решаемых при планировании, является удаление из F неиспользуемых в решении задачи операций. Если операция не вычисляет ни одну из переменных **W**, её не должно быть в плане.

Другой принципиально важный момент, связанный с планированием, заключается в разрешении конфликтов при возможности вычисления одних переменных разными путями. Поскольку структура ВМ может описывать разные предметные области, вводится понятие интерпретации ВМ, включающее отображение между входными значениями каждой операции и её выходными

значениями. В данной работе рассматриваются только такие интерпретации, при которых для любого означивания множества **V**, любой способ вычисления переменной y даёт одинаковый результат. При наличии в плане повторяющихся вычислений будут затрачиваться дополнительные ресурсы. Однако по такому плану можно отдельно выполнять выбор оптимального пути, что будет рассмотрено далее в этом разделе. Поэтому все операции, вычисляющие **W** по **V**, должны попадать в план.

Таким образом, планирование можно свести к обходу ориентированного двудольного графа, задаваемого отношениями **In** и **Out**. Наличие счётных множеств вхождений массовых операций делает невозможным обход в ширину, используемый при планировании на простых ВМ. Поэтому алгоритм планирования на ВММ должен эффективно работать с массовыми операциями, определяя вычислимые множества без прямого обхода всех вершин графа. Скорость работы должна зависеть не от размеров используемых массивов, а лишь от общего числа переменных и операций (и их сигнатур) в ВМ.

Ещё одним требованием является поддержка практически значимых задач. У планировщика должна быть возможность удобно описать вычислительную модель и задачу, интуитивно понимая как они должны выглядеть для конкретной предметной области. Конечно, оценить выполнение данного требования сложно.

При такой постановке задачи планирование фактически задаёт множество планов решения задачи, и в дальнейшем для него будет использоваться термин *препланирование*, когда важно разделить препланирование и получение не избыточного плана.

2.2 Идея решения

В данной работе предлагается идея планирования в несколько этапов. Основное отличие от других работ заключается в двойной постановке задачи на вычислительной модели на этапе препланирования.

Для каждого этапа предлагается алгоритм. Ключевое отличие данной работы заключается в уровне детализации алгоритмов планирования.

В алгоритме используется специальная модель, которая позволяет представлять в памяти ЭВМ счётные множества. Данное представление может быть использовано и самостоятельно в других задачах.

В подразделах 2.3–2.5 приводится подробное описание решения по идеям и компонентам.

2.3 Планирование в несколько этапов

Задача на вычислительных моделях предполагает конечные множества **V** и **W**. Однако при планировании не удобно работать с конечными множествами, поскольку они всё равно могут быть достаточно большими, а при наличии рекуррентных соотношений даже конечное входное множество переменных задаёт счётное множество вычислений.

Поэтому предлагается дополнительно задавать *общую* задачу на вычислительной модели. Она содержит два множества, являющихся надмножествами соответственно для **V** и **W** всего класса похожих задач. *Частная* задача уже должна задавать конечные множества входных и выходных переменных. Такой подход предполагает, что все компоненты массива являются однородными по свойствам объектами предметной области.

В таком случае часть задачи планирования – исключение заведомо лишних операций – можно выполнить для *общей* задачи, а определить конкретные диапазоны вхождений уже для *частной* задачи. Так можно оптимизировать планирование при решении некоторого множества задач, где меняется только количество объектов.

Как было сказано ранее, в результате планирования получается множество вхождений операций, необходимых для решения задачи. Эти вхождения используют переменные согласно своей сигнатуре. Поэтому результат можно представить в виде некоторой вычислительной модели, являющейся подмножеством исходной.

Планирование на частной и общей задачах решает задачу препланирования. На финальном этапе планирования необходимо выбрать

операции, не совершающие избыточных вычислений, возможно оптимизируя выбор основываясь на дополнительно заданных значениях переменных, или некоторым спецификациям операций и системы исполнения [11]. Подобные оптимизации не рассматриваются в текущей работе. Для выбора пути предлагается использовать следующий принцип случайного выбора: выполнять только одну (выбранную произвольным образом) из операций, вычисляющих требуемую переменную, а другие варианты выполнять только тогда, когда они требуются для вычисления других транзитивно требуемых для вычисления **W** переменных.

2.4 Способ представления счётных множеств

2.4.1 Описание подхода к представлению

Главная трудность разработки алгоритма, оперирующего счётными множествами, – хранение таких множеств в памяти ЭВМ. В данной работе предлагается вариант представления, а также алгоритмы осуществления множественных операций.

Удобнее всего представлять счётные множества в виде арифметических прогрессий, ограниченных или бесконечных. Для задания последовательности элементов достаточно знать начальный элемент, шаг прогрессии, а также, при необходимости, ограничение в виде последнего элемента или количества элементов. Последние два параметра можно однозначно вычислить друг из друга. Такие последовательности далее будут называться *диапазонами*.

Таким образом любое допустимое множество можно представить в виде набора диапазонов – ограниченных и неограниченных. При этом этот набор не должен зависеть от порядка добавления элементов. Например, две последовательности – чётных и нечётных чисел – должны равняться одной последовательности из всех натуральных чисел (с шагом один). Это необходимо для упрощения сравнения множеств, а также для оптимизации вычислений путём сокращения числа рассматриваемых объектов. Подобные преобразования

необходимо выполнять при вставке и удалении элементов, а множественные операции должны использовать эти базовые операции.

При добавлении диапазона находится пересечение с каждым диапазоном множества. В случае непустого пересечения находится разность диапазонов, и функция добавления должна вызываться для каждого нового диапазона этой разности. Рекурсия завершится на этапах добавления полностью уникальных диапазонов.

Удаление диапазона также можно выполнять рекурсивно. Для этого аналогично находится пересечение с каждым диапазоном множества. В случае непустого пересечения удаляется весь исходный диапазон, а все диапазоны разности между ним и пересечением добавляются в множество.

Задача нахождения пересечения двух диапазонов, а также их разности эквивалентна задаче китайской теоремы об остатках, поэтому для решения можно использовать расширенный алгоритм Евклида [18]. Стоит отметить, что пересечение – всегда один диапазон, а разность – это множество диапазонов.

После любой вставки диапазона в множество, необходимо выполнять попытку сокращения множества диапазонов. Когда конкретный элемент диапазона может быть включён в другие диапазоны, либо образовать новый, необходимо использовать систему приоритетов, учитывающую длину, шаг, количество элементов диапазона, и количество самих диапазонов. Были выбраны следующие принципы (в порядке убывания приоритета):

а) Неограниченные диапазоны при возможности заменяются диапазонами с меньшим шагом.

б) Неограниченный диапазон включает в себя максимальное количество элементов.

в) Ограниченные диапазоны не должны пересекаться.

г) Каждый элемент образует последовательность с ближайшим элементом, если для того нет более близкого элемента.

Полный алгоритм приведён в приложении А.

2.4.2 Описание расширения для многомерного случая

В работе также рассматривается модификация представления счётных множеств, поддерживающая многомерные объекты некоторого вида. Основная идея – хранить не множество диапазонов, а словарь с ключами в виде диапазона, и значениями в виде другого множества. В таком случае правила для сокращения представления учитываются для каждого множества ключей отдельно при условии равенства значений.

Правила сокращения обеспечивают универсальное представление, благодаря которому можно быстро сравнить два множества. При условии такого сокращения в пределах каждой размерности, это свойство будет сохраняться для каждой размерности.

На каждом уровне $i = 1,2, \dots, n$ используется следующий алгоритм вставки элемента $x = [x_1, x_2, \dots, x_n]$:

1) Для каждого диапазона y на текущем уровне множества:

1.1) Вычисляется $s = x_i \cap y$.

1.2) Если $s \neq \emptyset$:

1.2.1) y заменяется на диапазон пересечения, и диапазоны разности. Значение каждого диапазона равно исходному.

1.2.2) Добавить в множество $x' = [x_1, \dots, x_{i-1}, s, x_{i+1}, \dots, x_n]$.

1.2.3) Для каждого диапазона r в $(x_i - s)$: добавить в множество $x' = [x_1, \dots, x_{i-1}, r, x_{i+1}, \dots, x_n]$.

1.3) Иначе ($s = \emptyset$):

1.3.1) Добавить на текущий уровень диапазон x_i, а следующие компоненты x к его значению.

1.3.2) Выполнить сокращение диапазонов текущего уровня для x_i.

Удаление элементов происходит аналогичным образом.

В словаре последнего измерения значения всегда считаются равными. Таким образом при хранении одномерных элементов, представление будет полностью аналогично представлению, описанному в предыдущем пункте.

Конечно, такой алгоритм подразумевает, что измерение с меньшим номером имеет больший приоритет. Также такое представление не может выразить последовательности вида $\{(1,1),(2,2),(3,3),\dots\}$, которые могут возникнуть в процессе планирования. Поэтому данный способ накладывает много ограничений на алгоритм планирования и класс решаемых задач, однако всё ещё может показывать удовлетворительные результаты в некоторых случаях.

Пример автоматического сокращения многомерных последовательностей представлен на рисунке 2.1.

python	txt
`1 s = VS([('x',[R(0,200,2),R((1,10))])])`	`# S = {x[0,2,4,…,198][10,11,12,…]}`

`2 print(s)`	`{`
	`  x: {0,2,...,198}(step=2):`
	`  Range(1,10)`
	`}`

`3 s.add(('x',[R(1,200,2),R((2,10))]))`	`# Добавить x[1,3,5,…,199][10,12,14,…]`

`4 print(s)`	`{`
	`  x: {0,2,...,198}(step=2):`
	`  Range(1,10),`
	`  {1,3,...,199}(step=2):`
	`  Range(2,10)`
	`}`

`5 s.add(('x',[R(1,200,2),R((2,11))]))`	`# Добавить x[1,3,5,…,199][11,13,15,…]`

`6 print(s)`	`{`
	`  x: {0,1,...,199}(step=1):`
	`  Range(1,10)`
	`}`
	`# S == {x[0,2,3,…,199][10,11,12,…]}`

Рисунок 2.1 – Пример сокращения многомерных последовательностей

В данном примере используется синтаксис программной реализации из главы 3 настоящей работы. При последнем добавлении диапазоны во втором измерении объединились в один, поскольку совпадали также их первые измерения. Далее вторые измерения у двух последовательностей стали равны, и они сократились по первому, став единым диапазоном.

2.5 Описание алгоритма планирования

В подразделе описывается модификация алгоритма, описанного в [7]. Модификация предназначена для планирования на общих и частных задачах. Также текущая работа переводит алгоритм из описания в рамках теории

множеств в удобный для реализации на языках программирования вид, а также предлагает более эффективную работу алгоритма для ряда случаев за счёт разделения общей и частной задач.

2.5.1 Общее представление входных данных

Вычислительная модель представлена двумя словарями – для переменных и операций, – в которых доступно получение элемента по имени за $O(1)$. В дальнейшем предполагается, что все свойства операций и переменных можно получить из этого словаря за константное время. Конечно, в реальности такой структурой чаще всего будет хэш-таблица, и при большом количестве объектов сложность поиска будет стремиться к $O(\log(n))$.

Для каждой переменной определены два списка операций, использующих её как входную или выходную переменную соответственно. Для операций же определены списки входных и выходных переменных. При этом, для используемых компонент массивов заданы линейные функции от номера вхождения. Конечно, в простых операциях линейный коэффициент всегда равен нулю.

2.5.2 Планирование на общей задаче

На данном этапе планирования на вход алгоритму требуется исходное представление вычислительной модели и множества **V** и **W** общей задачи. На этом этапе происходит *фильтрация* ВМ – удаляются записи о переменных и операциях из её внутреннего представления. Также на этом этапе образуются *пси-операции*. Это особая операция, которая задаёт параметрический терм. То есть пси-операция задаёт счётные диапазоны вхождений операций ВМ. Для пси-операции заранее определены входные переменные, необходимые для выполнения любого вхождения из задаваемого ею множества вхождений. Выходное множество переменных пси-операции также заранее определено описываемыми множествами вхождений. Таким образом пси-операция имеет свойства простой операции, но может связывать счётные множества переменных.

Алгоритм состоит из двух этапов – *восходящей* и *нисходящей* частей. В восходящей части выполняется последовательное определение достижимых из V операций. Когда в очередное множество попадают вхождения массовых операций, начинается алгоритм имитации планирования, который строит пси-операции. В нисходящей части происходит выбор из отфильтрованных или сгенерированных восходящей частью операций, которые участвуют в вычислении W. Детализированное описание алгоритма разбито по указанным частям, а также отдельным логическим блокам.

2.5.2.1 Восходящая часть планирования

1) Создаются множества:

V_i – изначально копия V – множество достигнутых за все шаги переменных;

F_i – пустое множество операций текущего шага;

F_{sum} – пустое множество достигнутых операций.

2) Все простые операции, у которых вход – пустой список, добавляются в F_{sum}, все их выходные значения добавляются в V_i.

3) Создаётся множество V_{new} – изначально копия V_i – множество достигнутых на данном шаге переменных;

4) Выполняется бесконечный цикл до выполнения одного из условий выхода.

4.1) Очищаются множества F_i и F_j'

4.2) Создаётся пустое множество имён ops. Для каждого уникального имени переменной в V_{new} к ops добавляются имена всех операций, использующих эту переменную.

4.3) Для каждого on имени операции в ops:

4.3.1) Если on – простая операция, её нет в F_{sum} и каждая используемая ей переменная содержится в V_i, она добавляется в F_i. Перейти к следующему имени в (4.3).

4.3.2) Если on – массовая операция, то вычисляется S всех диапазонов применимости операции при помощи *Алгоритма определения применимости операции к множеству переменных* V_i.

4.3.3) К F_i' добавляется разность S и F_{sum}.

4.3.4) Перейти к следующему имени в (4.3).

4.4) Если F_j' не пустое, выполнить *Алгоритм имитации планирования*.

4.5) Если F_i пустое, восходящая часть завершается. Результат работы представлен множествами F_{sum} и V_i. [ВЫХОД]

4.6) F_i добавляется к F_{sum}.

4.7) V_{new} очищается.

4.8) Для каждого элемента в F_i добавить в V_{new} соответствующие выходные переменные простой операции (включая пси-операции) или конечной последовательности вхождений массовой операции.

4.9) V_{new} добавляется к V_i.

2.5.2.2 Алгоритм имитации планирования

Ранее вычислены: V_i, F_i, F_j'.

Результатом является пополнение F_i операциями (включая пси-операции, сконструированные при имитации планирования), достигаемыми для исходного F_j'. Также в результате работы изменяется представление вычислительной модели исходя из образованных пси-операциями связей.

1) Создаются пустые множества V_{new}', F_{sum}'; пустой список P новых пси-операций; множество V_j', равное V_i; пустой словарь L.

2) Выполняется бесконечный цикл до выполнения одного из условий выхода.

2.1) Очищается V_{new}'; создаются L', равный L, и пустое множество R для подлежащих удалению вхождений операций.

2.2) Для каждой бесконечной последовательности вхождений каждой массовой операции из F_j' выполняется *Алгоритм попытки построения пси-операции*.

2.3) Для каждой конечной последовательности вхождений каждой массовой операции из F_j' выполняется *Алгоритм попытки построения пси-операции*.

2.4) Из F_{sum}' и F_j' удаляются все вхождения, присутствующие в R.

2.5) F_j' добавляется в F_{sum}'.

2.6) Для каждого on имени в R:

2.6.1) Из области определения операции убираются все вхождения R.

2.6.2) Если on находится в L, то из L удаляются все вхождения операции, находящиеся в R.

2.7) Для каждого элемента в F_j':

2.7.1) Если это простая операция, добавить её в V_{new}'.

2.7.2) Если это последовательность (может быть только конечной) вхождений массовой операции из F_j', добавить в V_{new}' соответствующие ей выходные переменные согласно определению операции в ВМ.

2.8) Очистить F_j'.

2.9) Создаётся пустое множество имён ops. Для каждого уникального имени переменной в V_{new}' к ops добавляются имена всех операций, использующих эту переменную.

2.10) Для каждого on имени операции в ops:

2.10.1) Если on – простая операция, она добавляется в F_i. Перейти к следующему имени в (2.10).

2.10.2) Если on – массовая операция, то вычисляется множество S всех диапазонов применимости операции при помощи *Алгоритма определения применимости операции к множеству переменных* V_i'.

2.10.3) К F_j' добавляется разность S и F_{sum}'.

2.10.4) Перейти к следующему имени в (2.10).

2.11) Если F_j' пустое, добавить весь F_{sum}' в F_i и перейти к (3), иначе начать новую итерацию (2).

3) Для каждой операции в P вычисляются её входы и выходы, и операция добавляется в словарь операций ВМ как простая операция. Для каждого уникального имени входных и выходных переменных операции эта операция указывается соответственно в соответствующие поля этой переменной в словаре переменных ВМ.

2.5.2.3 Алгоритм определения применимости операции к множеству переменных

Вход: массовая операция op (со своим множеством определения), множество переменных V_s.

Результат: множество вхождений операции, которое может быть применено к данному множеству переменных.

1) Создаётся множество S, изначально равное области определения операции.

2) Для каждой входной переменной iv согласно сигнатуре операции:

2.1) Если это простая переменная, или компонента массива, не зависящая от номера вхождения, и она не содержится в V_s, вернуть пустое множество.

2.2) Далее iv – компонента массива, зависящая от номера вхождения.

2.3) Создаётся пустое множество S_{iv}.

2.4) Для каждой последовательности r переменной iv в V_s:

2.4.1) При помощи расширенного алгоритма Евклида находится пересечение последовательности, задаваемой iv, и r.

2.4.2) Из полученной последовательности обратным линейным преобразованием легко получить последовательность индексов вхождений операции. Если пересечение существует, она добавляется в S_{iv}.

2.4.3) S обновляется пересечением с S_{iv}.

2.5) В результате в S остались последовательности, применимые по всем входным переменным, если в некоторый момент в (2.4) было получено пустое множество, можно сразу завершить текущий алгоритм и выдать пустой результат.

2.5.2.4 Алгоритм попытки построения пси-операции

Вход: последовательность op вхождений массовой операции.

Ранее вычислены: L', L , V'_{new}, V'_j, F_i, R, P.

В результате работы алгоритма могут измениться множества R, P, L , V'_{new}, F_i. Напрямую возвращаемый результат отсутствует.

1) Если последовательность op конечная, она разделяется на элементы, и для каждого элемента s выполняются следующие шаги:

1.1) Если s содержится в R, перейти к следующему элементу в (1).

1.2) Если имя операции op не содержится в словаре L, добавить в L запись с именем операции со списком из единственного элемента s. Для каждого вхождения сохраняется текущий уровень j. В этом случае перейти к следующему элементу в (1).

1.3) Все вхождения e из L под именем операции op сортируются в порядке неубывания $|j(s) - j(e)|$, где $j(s)$ – номер вхождения s, $j(e)$ – номер вхождения e.

1.4) Для каждого вхождения e в отсортированном списке выполняется:

1.4.1) Строятся все возможные идентичные термы с помощью *алгоритма построения идентичных термов* для корневых вхождений s и e.

1.4.2) Для каждой пары термов выполняется проверка *самоподпитываемости* терма. Пара термов задаёт шаг для каждого своего вхождения операций, а таже шаг для используемых переменных. Легко найти то множество переменных, которые не вычисляются на предыдущих применениях терма. Если это множество находится в V_j' создаётся пси-операция, называемая *последовательной*. Она добавляется в P и в F_i, а все вхождения операций, задаваемые этой пси-операцией, добавляются в R. В V_{new}' добавляются все вычисляемые пси-операцией переменные.

1.5) Если пси-операция создана не была, s с соответствующим уровнем вхождения сохраняется в L.

2) Если последовательность op бесконечная:

2.1) Находится пересечение с R и, если оно не пустое, выполняется текущий алгоритм для каждой последовательности из разности op и R. Дальнейшие шаги не выполняются в рамках текущего алгоритма.

2.2) Если пересечение пустое, вся такая последовательность образует пси-операцию, называемую *автоматической*. Она добавляется в P и в F_i, а все

вхождения операций, задаваемые этой пси-операцией, добавляются в R. В V'_{new} добавляются все вычисляемые пси-операцией переменные

2.5.2.5 Алгоритм построения идентичных термов

Вход: два вхождения массовой операции $fl - s$ и e. $j(e)$ и $j(s)$ – номера этих вхождений.

Выход: список бесповторных термов с корнями s и e.

Ранее вычислены: L', R, P.

Также здесь используется множество означиваний C, которое изначально пустое, а при рекурсивном вызове построения термов туда помещается информация, необходимая построения бесповторных термов.

1) Если e содержится в R, результатом работы является пустой список термов. [ВЫХОД]
2) Если неограниченный диапазон с шагом $(j(e) - j(s))$ и начальным элементом $j(s)$ не содержится полностью в области определения операции op, результатом работы является пустой список термов. [ВЫХОД]
3) Если $j(s) < j(e)$, перейти к (7).
4) Создаётся список $terms$ пустых списков для каждого входа в сигнатуре операции.
5) Для каждого входа iv в сигнатуре операции выполняется:
   5.1) Если iv – простая переменная, или массовая переменная, не зависящая от номера вхождения (линейный коэффициент равен 0), добавить в соответствующий список в $terms$ (iv, iv), перейти к следующему входу в (5).
   5.2) Далее iv – компонента массива, зависящая от индекса операции. Вычисляются номера i_1^{iv}, i_2^{iv}, соответствующих компонент для номеров вхождения s и e соответственно.
   5.3) Определяется список операций, вычисляющих переменную iv. Для каждой операции соответствующий ей список в L' упорядочивается по возрастанию уровня.

5.4) Для каждой пары различных вхождений f_1 и f_2 операции fl, таких что уровень f_1 больше или равен уровню f_2 (алгоритм перебора неупорядоченных наборов из двух элементов):

5.4.1) Если хотя бы одно вхождение (f_1, f_2) использовалось в рекурсивной ветке для вычисления iv (в C содержится запись (f_1, iv) или (f_2, iv)), перейти к другой паре в (5.4).

5.4.2) Проверяются все выходные переменные fl, и если f_1 вычисляет i_1^{iv}, а f_2 – i_2^{iv}, то попытка построения продолжается, иначе перейти к другой паре в (5.4).

5.4.3) В текущее множество означиваний C добавляются записи для операции fl: конкретные вхождения f_1 и f_2 вычисляют iv.

5.4.4) Запускается *алгоритм построения идентичных термов* уже для f_1 и f_2. Результат добавляется к соответствующему текущей переменной списку в $terms$.

5.4.5) Множество C возвращается в исходное состояние.

5.5) Если список в terms пустой, он заполняется термом, равным самой входной переменной iv.

6) Строится список всех упорядоченных комбинаций внутренних термов для входных переменных корневых вхождений s и e. Каждая комбинация вместе с корневой операцией задают два идентичных терма максимальной глубины из L'. Каждая пара термов добавляется к результату.

7) Если $j(s) \leq j(e)$, выполнить пункты (4) – (6), только (5.4) выполнять для пар, когда уровень f_1 меньше или равен уровню f_2.

2.5.2.6 Нисходящая часть планирования

Данный алгоритм не требует дополнительного подробного описания. На отфильтрованной в восходящей части ВМ происходит обход в обратном направлении (начиная с **W**), причём никакие специальные случаи рассматривать не нужно, поскольку вхождений массовых операций конечное (и относительно небольшое) число, а пси-операции в плане применимости аналогичны простым операциям. Все действия полностью аналогичны восходящей части. Стоит

отметить лишь, что аналог *алгоритма определения применимости операции к множеству переменных* здесь должен объединять, а не пересекать промежуточные результаты, поскольку важна применимость для вычисления хотя бы одного значения. Результатом работы алгоритма являются множества достигаемых переменных и операций, что и является планом для общей задачи.

2.5.3 Планирование на частной задаче

На данном этапе планирования на вход алгоритму требуется исходное представление вычислительной модели, множества **V** и **W** частной задачи, а также результат планирования на общей задаче. Этот результат представляет частично отфильтрованную ВМ со сконструированными пси-операциями. На текущем этапе необходимо вернуться от специфического представления вхождений операций к исходным массовым операциям, используя заданные в задаче конечные диапазоны переменных.

Поскольку пси-операции были образованы в восходящей части, они имеют нижнюю границу применимости. Поэтому для ограничения с обратной стороны, планирование следует начать с нисходящей части. На этом этапе будут определены границы применимости потенциально используемых пси-операций, а в восходящей части будет производиться дополнительная фильтрация диапазонов вхождений массовых операций при раскрытии пси-операций.

2.5.3.1 Нисходящая часть планирования

Здесь выполняется обход полностью аналогичный нисходящей части планирования на общей задаче, кроме случаев с пси-операциями. Для пси-операции нужно найти множество индексов применений описываемого параметрического терма, при которых вычисляется что-то из текущего множества переменных. Для этого достаточно рассмотреть последовательности вхождений массовых операций, задаваемых пси-операцией, и объединить их применимости, найденные согласно алгоритмам, описанным ранее.

Далее действия отличаются в зависимости от типа пси-операции. Автоматические пси-операции могут применяться в произвольном порядке. А вот последовательная пси-операция на каждом шаге использует полученные на

предыдущих шагах переменные, поэтому требует последовательного исполнения начиная с первого описываемого терма. Поэтому множество применимости последовательной пси-операции расширяется от максимального применения до нуля. При повторной обработке пси-операции множества применимости объединяются в одно, которое будет использовано в восходящей части.

Имея множество применимости пси-операции легко вычислить все используемые таким образом вхождения массовых операций. Результат добавляется к текущему множеству операций, что породит новые зависимости на следующем шаге нисходящего обхода. Полученное таким образом множество не представляет ценности и нужно лишь для упрощения обхода, а главным результатом алгоритма являются ограничения, вычисленные для пси-операций.

2.5.3.2 Восходящая часть планирования

На этом этапе алгоритм похож на алгоритм восходящей части планирования на общей задаче. Однако не выполняется имитация планирования, а все вхождения массовых операций сразу попадают в множество F_i. Также алгоритм отличается обработкой пси-операций. Автоматические пси-операции никогда не будут применимы, поскольку множество переменных в этом процессе остаётся конечным. Для последовательных пси-операций в нисходящей части определён диапазон применимости, достаточный для вычисления W, поэтому при возможности применить такую операцию, к текущему множеству операций добавляются все вхождения из этого диапазона.

Результатом работы алгоритма являются множества достигаемых переменных и операций, что является планом и может быть использовано интерпретатором или другим планировщиком исполнения.

2.6 Анализ решения

2.6.1 Ключевые особенности решения

Предложенный подход с планированием на общей и частной задачах позволяет задать ограничения на вычисления. Иначе это можно сделать,

расширив определение ВММ, добавив переменные-счётчики и предикаты, однако очевидно, что алгоритм планирования в таком случае требует расширений. В то время как реализация алгоритма для простых ВММ позволяет проверить на практике применимость подхода в целом.

Другим важным результатом является разработка способа хранения множеств элементов. Особенность заключается в автоматическом выделении последовательностей по некоторым простым правилам. Даже если правила будут изменены, сам принцип имеет хороший потенциал для использования при работе с ВММ.

В отличие от других решений, здесь разрешается использование операций с пустыми входными множествами. Для них предусмотрена обработка в начале алгоритма, и из выход автоматически добавляется к множеству **V**. Это позволяет сделать описание более удобным, поскольку появляется возможность использования констант предметной области без их прямого описания пользователем для каждой задачи на ВМ.

2.6.2 Оценка корректности

Одним из требований к решению была корректность выводимого плана. В план должны попасть те и только те операции, которые участвуют в вычислении переменных из **W** (при заданном множестве **V**).

Сам принцип обхода графа зависимостей обеспечивает корректность достигаемых вершин, поскольку полностью имитирует реальное последовательное вычисление. На каждом шаге в множество операций попадают только те, которые используют только переменные текущего достигнутого множества. А переменные в него попадают только те, которые вычисляются операциями предыдущих шагов. При этом очевидно, что для каждой операции, которая должна попасть в план, существует шаг, когда в множество попадут все используемые переменные.

Условие самоподпитываемости для пси-операций также обеспечивает возможность реального вычисления любого из описываемых ими вхождений.

2.6.3 Оценка алгоритмической сложности

Другим важным требованием к решению является независимость скорости работы алгоритма от размеров используемых в задаче массивов. Полное сокращение множества переменных до конечного числа зависимостей происходит на этапе планирования на общей задаче. При наличии закономерности в применении операций, она выявится не позднее, чем появится по два вхождения каждой операции описываемого пси-операцией терма. Очевидно, что при используемом обходе это произойдёт за время, зависящее только от сложности описываемых отношений.

На этапе планирования на частной задаче всё сводится к обходу определённого конечного числа операций, включая пси-операции. При раскручивании пси-операций до вхождений, последние добавляются непрерывными последовательностями. Это позволяет при обходе вычислять следующий шаг массовым образом без отдельной обработки каждого элемента последовательности.

2.6.4 Недостатки

Главным недостатком предлагаемого решения является неэффективность при использовании ограниченных массивов. Формулирование общих задач предполагает бесконечное множество однородных компонент, которое ограничено только с одной стороны. Однако ряд задач и предметных областей использует такие массовые описания, которые невозможно расширить до общего случая. Например, клеточно-автоматное моделирование. Там используются специальные отношения для граничных вычислений, что не позволяет выделить счётные множества элементов. С другой стороны, такой класс задач требует также использования многомерных массивов. Поэтому этот момент следует учесть при переходе к многомерному алгоритму.

Также предпринималась попытка введения многомерных массивов сразу в текущую версию алгоритма. Однако полученное представление покрывает не все классы многомерных последовательностей. Поэтому было решено

отказаться от многомерных задач на данном этапе работы. Для расширения алгоритма необходимы дальнейшие исследования.

Другой недостаток был выявлен на поздних этапах исследования – неэффективность алгоритма сокращения множества элементов, описанного в приложении А, для ряда случаев. Предложенные приоритеты конфликтуют и добавленные ограничения позволяют получить некоторое представление только после нескольких итераций объединения и разделения последовательностей.

3 Программная реализация системы планирования

В разделе описывается программный комплекс из двух компонентов: библиотеки планирования и сервиса, предоставляющего API для планирования задач на вычислительных моделях. Также приводятся результаты тестирования первого компонента для некоторых реальных задач численного моделирования.

3.1 Описание библиотеки планирования

3.1.1 Архитектура

Для реализации алгоритма выбран язык Python из-за динамической типизации, автоматического управления памятью, использовании высокоуровневых структур данных, и поддержки объектно-ориентированного программирования [19]. Эти требования возникают из-за необходимости оперировать сложными объектами в используемых алгоритмах.

Система планирования реализована в виде библиотеки для языка Python. Описаны классы для используемых объектов, которые необходимо сконструировать пользователю библиотеки. Также библиотекой предоставляются процедуры для планирования отдельно на общей и частной задачах. Описание программы и руководство программиста расположены в приложениях Б и В соответственно.

Библиотека разделена на модули (файлы), представленные в таблице 1.

Таблица 1 – Список модулей библиотеки планирования

Файл	Размер, строк	Описание
alglogger.py	34	Работа с протоколированием в ходе работы.
base_math.py	32	Вычисление НОК, НОД. Вариант расширенного алгоритма Евклида.
fit_tools.py	231	Вспомогательные процедуры при планировании на частной задаче.
json_fields_names.py	31	Описание полей JSON-формата ВМ.
model.py	238	Описание классов объектов ВМ.

Продолжение таблицы 1

Файл	**Размер, строк**	**Описание**
plan.py	238	Основные (высокоуровневые) процедуры планирования.
plan_tools.py	287	Вспомогательные процедуры при планировании на общей задаче.
ranges.py	246	Класс диапазонов.
terms.py	82	Работа с термами при построении пси-операций.
variables_set.py	638	Классы множеств для хранения диапазонов.

Зависимости между модулями библиотеки, а также взаимодействие с пользователем представлены на рисунке 3.1.

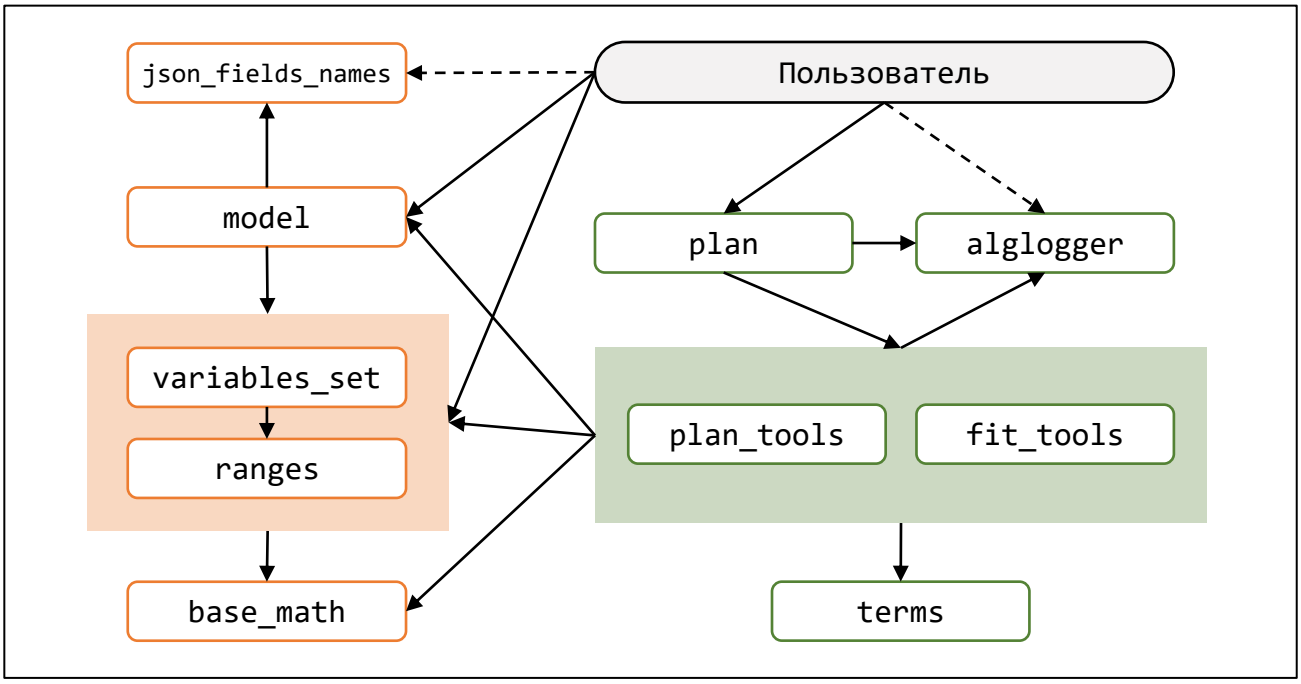

Рисунок 3.1 – Схема взаимодействия модулей библиотеки планирования

3.1.2 Ключевые классы

В данном пункте описываются особенности реализации классов библиотеки и процедур планирования.

Диапазон (класс *Range*).

Атрибуты диапазона представлены объектами класса. Причём одинаковое представление используется как для ограниченных, так и для неограниченных диапазонов. Ограниченность определяется по параметру «количество

элементов» – для не ограниченных он равен нулю. Для удобства методы пересечения и вычитания двух диапазонов переопределены для операторов «умножить» и «минус» соответственно. При этом оператор «минус» является итератором по всем диапазонам результирующего множества.

Множество диапазонов (класс *DimScope*).

Реализованы методы добавления и удаления диапазонов, а также сокращение множества по алгоритмам, описанным ранее в работе. Диапазоны хранятся в структуре «множество», чтобы ускорить время поиска и удаления элементов.

Данный класс не используется напрямую из алгоритма и только используется в следующем классе.

Множество (класс *VariablesSet*).

Содержит «множество» простых имен, а также «словарь» (имя : множество диапазонов). Поддерживает добавление и удаление как конкретных элементов, так и полных записей об имени в множестве. Реализованы изменяющие и копирующие операции объединения, пересечения, вычитания множеств. Также реализован итератор по множеству. Пример такого объекта с комментариями представлен на рисунке 3.2.

python

```python
V= VariablesSet(
  [
    ('x', [Range(1)]),
    # x[1]

    ('z', [Range(1,10,2)]),
    # z[1], z[3], z[5], z[7], z[9]

    ('y', [Range((2,0))]),
    # y[0], y[2], y[4], ...

    ('y', [Range((3,0))])
    # y[0], y[3], y[6], ...
  ]
).difference( # разность с другим множеством
  VariablesSet([
    ('y', [Range((6,10))]
    # y[10], y[16], y[22], ...
  ])
)
```

txt

```
Реальный объект V (после
сокращения представления):

{
x: {1};
z: {1,3,...,9}(step=2);
y: Range(3,0),
   {4},
   Range(6,2)
}

Расшифровка диапазонов:

x[1]
z[1], z[3], z[5], z[7], z[9]
y[0], y[3], y[6], ...
y[2], z[8], z[14], ...
y[4]
```

Рисунок 3.2 – Пример описания объекта класса VariablesSet

Как видно, каждый диапазон оборачивается в массив, так как синтаксис подразумевает использование многомерных случаев.

Операции.

Представлены двумя классами для простых (*SOperation*) и массовых (*MOperation*) операций. Содержат два списка входных и выходных переменных. Списки содержат имена простых переменных, либо кортежи с именем массива и кортежем (a, b) линейной функции использования. Для простых операций хранятся флаги пси-операции и её типа, а для массовых – множество применимости. Множество применимости содержит единственное имя – саму операцию.

Переменные.

Представлены двумя классами для простых (*SVariable*) и массовых (*MVariable*) переменных. Классы содержат два списка использующих переменную операций – как входную и как выходную. Операции в них всегда представлены только именем операции.

Вычислительная модель (класс *Model*).

Содержит два словаря для переменных и операций. Ключами являются имена, а значениями – объекты описанных выше классов.

Функция планирования на общей задаче (*model_plan*).

Принимает объект Model и два объекта VariablesSet в именованных параметрах V и W для соответствующих множеств задачи. Возвращается копия исходной ВМ с новыми пси-операциями, а также с отфильтрованными в ходе работы множествами переменных и операций.

Функция планирования на частной задаче (*model_fit*).

Принимает два объекта Model для исходной ВМ и результата работы model_plan, и два объекта VariablesSet в именованных параметрах V и W для соответствующих множеств задачи. Возвращается копия исходной ВМ с отфильтрованными в ходе работы множествами переменных и операций.

3.1.3 Особенности реализации

Поскольку алгоритм описан достаточно подробно, сложностей в его реализации в виде процедур языка Python не возникает.

В алгоритме используются сложные структуры данных, и при их копировании (например, при копирующих множественных операциях) необходимо выполнять глубокое копирование, предоставляемое стандартной библиотекой *copy*, – функция *deepcopy*.

Поскольку конструируемым пси-операциям необходимо давать имя при добавлении в структуру ВМ, используется глобальная переменная-счётчик, которая увеличивается после каждого использования. Имя конструируется по шаблону с использованием текущего значения счётчика.

При построении идентичных термов используется сочетание всех вариантов для термов входных использований. Его можно получить, используя функцию *product* стандартной библиотеки *itertools*.

Часто в структурах данных используются одновременно и простые имена для простых, и кортежи для массовых случаев, а также объекты разных предоставляемых библиотекой классов. Удобно использовать функцию *isinstance* в качестве условия для разделения ветвей обработки.

3.2 Описание сервиса планирования

В подразделе описывается способ применения разработанной библиотеки планирования.

3.2.1 Использование в системе LuNA

В лаборатории синтеза параллельных программ ИВМиМГ СО РАН разрабатывается система активных знаний LuNA, которая позволяет описывать и исполнять параллельные программы на основе вычислительных моделей [20]. В ней используется микросервисная архитектура, и планировщик там используется как один из этапов обработки задачи пользователя.

Для этой системы в ходе работы была реализована программа на языке Python, предоставляющая REST API планирования. Доступны отдельные HTTP-

запросы для планирования на простой вычислительной модели и на вычислительной модели с массивами. Для реализации серверного взаимодействия использовался фреймворк Tornado [21], поскольку позволяет просто описать запросы, работает асинхронно, а также включает в себя инструменты для автоматического запуска сервера.

На вход подаётся вычислительная модель и задачи (общая и частная для ВММ) в установленном в системе LuNA формате (описан далее). На выходе в том же формате получается вычислительная модель, содержащая результат препланирования.

3.2.2 JSON формат вычислительных моделей с массивами

Для удобства хранения ВММ и возможности интеграции в систему синтеза программ на ВМ был разработан JSON-формат описания ВММ.

JSON-объект вычислительной модели содержит два объекта – словари переменных (*variables*) и операций (*operations*), – а также поля с пользовательской информацией – имя (*name*) или описание (*description*). Ключами являются id переменных и операций, а значениями – объекты со свойствами операции или переменной. Для каждого объекта необходимо поле *type* с возможными значениями *simple* и *massive* для простых объектов и массовых соответственно. Также для каждого объекта допустимы поля с пользовательской информацией – имя (*name*), описание (*description*).

Для объекта массовой операции необходимо поле *scope*, которое задаёт область определения операции. *scope* представляет массив 3-х элементных массивов вида $[a, b, c]$, каждый из которых задаёт последовательность с шагом a, начиная с b. Элемент c здесь задаёт количество элементов, и, если c равно нулю, последовательность считается неограниченной. Например, объект на рисунке 3.3 задаёт множество $\{0,1,2,\ldots,48,49,51,53,\ldots,67,69,70,73,\ldots\}$.

Отношения между переменными и операциями задаётся в объектах операций – поля *inputs* и *outputs* для входных и выходных использований соответственно. Эти поля задают массив, в котором каждый объект содержит либо только поле *id* (для простых переменных), либо поля *id* и *scope* (для

массивов). *scope* является массивом вида $[a, b]$, что означает использование в i-ом вхождении операции $a \cdot i + b$ вхождения массива. При использовании элементов массивов в простых операциях, значение a должно быть равно нулю.

```json
"scope": [
  [1, 0, 50],
  [2, 51, 10],
  [3, 70, 0]
]
```

Рисунок 3.3 – Пример JSON описания множества диапазонов

На рисунке 3.4 приведён пример описания в предложенном формате вычислительной модели с двумя массивами x и y, простой переменной n и одной массовой операцией f, вычисляющей $y[i]$ по $x[i]$ и n для натуральных i.

```json
{
  "name": "пример ВМ",
  "variables": [
    { "id": "n", "type": "simple"  },
    { "id": "x", "type": "massive" },
    { "id": "y", "type": "massive" }
  ],
  "operations": {
    "f": {
      "type": "massive",
      "description": "вычисляет (x[i], n) -> y[i]",
      "inputs": [
        { "id": "n" },
        { "id": "x", "scope": [1, 0] }
      ],
      "outputs": [
        { "id": "y", "scope": [1, 0] }
      ],
      "scope": [[1, 1, 0]]
    }
  }
}
```

Рисунок 3.4 – Пример JSON описания вычислительной модели

3.3 Тестирование

3.3.1 Подход к тестированию

При тестировании алгоритма важно учесть два момента.

Во-первых, при планировании на общей задаче могут выделяться пси-операции двух видов. Получение последовательных пси-операции представляет наибольшую ценность алгоритма, а они возможны только при наличии некоторого вида рекуррентных соотношений.

Во-вторых, нужно проверить удобство использования подхода к планированию на задачах без рекуррентных соотношений. Разработанный подход позволяет ограничить вычисления для счётных множеств операций, что также необходимо проверить.

Итак, проверяются возможность планирования и его результат, удобство описания вычислительной модели и задач.

При тестировании составляется вычислительная модель предметной области некоторой задачи численного моделирования, а также формулируются общая и частная задачи. Оценивается удобство такого описания. Также оценивается количество совершённых алгоритмом действий (исходя из протокола работы) и корректность результата. Под корректностью здесь подразумевается соответствие результатам, полученным при планировании вручную. Для тестируемых случаев это проверить не сложно.

Целью планирования является доказательство применимости для некоторых классов практически значимых задач. В следующих пунктах рассматриваются две такие тестовые задачи, удовлетворяющие описанным требованиям к тестированию.

3.3.2 Задача сейсмической свёртки

В области обработки геофизических данных существует задача многоканальной свертки сейсмических сигналов [22]. Взаимокорреляционная свертка продолжительных по времени регистрируемых и опорных сигналов вычисляется при помощи быстрого преобразования Фурье. Причём используется секционирование длинных последовательностей, что позволяет применить

массовый подход при описании задачи в формате ВМ. Эффективность параллельной многоканальной свертки сейсмических сигналов рассмотрена в работе [23]. Там уже применялся подход, основанный на ВМ, однако не использовались ВММ, а каждый объект задачи представлялся в самостоятельном виде.

При тестировании не рассматривается построение графиков, а также организация загрузки данных. Основная операция корреляционной свёртки вычисляет сейсмограмму свёртки по файлу сеанса, опорному сигналу и интервалу. Для получения более разнородной ВМ, будем считать, что опорный сигнал и интервал постоянные для всех каналов. Также в работе [24] рассматривается вариант выполнения операции на графическом процессоре. Сигнатура операции будет совпадать. Базовая вычислительная модель для этой предметной области представлена на рисунке 3.5.

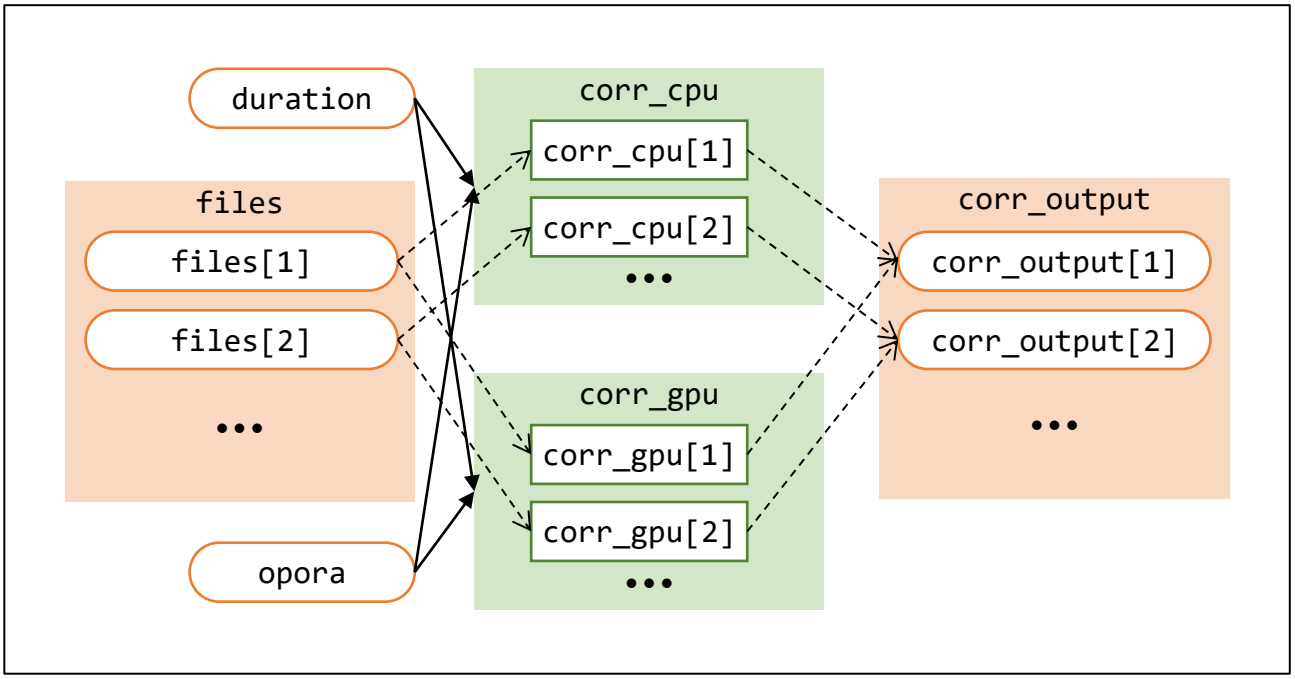

Рисунок 3.5 – Вычислительная модель потоковой сейсмической свертки

Общая задача на ВМ формулируется так: заданы все файлы сеансов (*files*), опорный сигнал (*opora*) и интервал (*duration*); требуется найти все сейсмограммы (*corr_output*). В частной задаче на ВМ указывается, что файлов и сейсмограмм используется по 100. При тестировании же использовалось усложнение, и файлов было предоставлено больше, чем требуется для

вычисления **W**, а именно 1000. Ожидалось автоматическое определение необходимого подмножества **V**.

Программа для решения этой задачи, использующая библиотеку планирования, может быть записана так, как показано на рисунке 3.6. Стоит отметить, что пользователем описываются только сигнатуры операций. В данном случае использование переменных в операциях указано вручную, хотя это может быть определено и автоматически по сигнатурам операций.

```python
# from {...} import *

# псевдонимы для сокращения записей
R = Range
VS = VariablesSet

modelOperations=[
    MOperation('corr_cpu',
               ins=[('files',(1,0)), 'opora', 'duration'],
               outs=[('corr_output',(1,0))],
               scope=VS([('corr_cpu',[R((1,0))])])
               ),
    MOperation('corr_gpu',
               ins=[('files',(1,0)), 'opora', 'duration'],
               outs=[('corr_output',(1,0))],
               scope=VS([('corr_gpu',[R((1,0))])])
               )
]

modelVariables=[
    SVariable('opora', entry=['corr_cpu','corr_gpu'], comp=[]),
    SVariable('duration', entry=['corr_cpu','corr_gpu'], comp=[]),
    MVariable('corr_output', entry=[], comp=['corr_cpu','corr_gpu']),
    MVariable('files', entry=['corr_cpu','corr_gpu'], comp=[])
]

seis_model = Model(v={x.label : x for x in modelVariables},
              o={x.label : x for x in modelOperations})

# планирование на общей задаче
seis_model_t1 = model_plan(seis_model,
                           V=VS([('files',[R((1,0))]), 'opora', 'duration']),
                           W=VS([('corr_output',[R((1,0))])])
                           )
# планирование на частной задаче
seis_model_t2 = model_fit(seis_model, seis_model_t1,
                          V=VS([('files',[R(0,1000)]), 'opora', 'duration']),
                          W=VS([('corr_output',[R(0,100)])])
                          )
```

Рисунок 3.6 – Использование библиотеки при планировании для задачи сейсмической свёртки

При запуске системы планирования сразу были сконструированы две автоматические пси-операции, которые независимо вычисляют все возможные сейсмограммы. Поскольку других операций в ВМ нет, планирование на общей задаче на этом завершилось. Результат (внутренне представление промежуточной ВМ) представлен на рисунке 3.7. Зелёным цветом выделены множества термов, описываемых пси-операциями.

```txt
{
    'opora': opora(entry:['psi0', 'psi1'], comp:[]),
    'duration': duration(entry:['psi0', 'psi1'], comp:[]),
    'corr_output': corr_output(entry:[], comp:['psi0', 'psi1']),
    'files': files(entry:['psi0', 'psi1'], comp:[])
}
psi0: psi0(psi(auto): corr_gpu[(1, 0)](files[(1, 0)], opora, duration))(
    ins:{
        duration; opora;
        files: Range(1,0)
    },
    outs:{
        corr_output: Range(1,0)
    })
psi1: psi1(psi(auto): corr_cpu[(1, 0)](files[(1, 0)], opora, duration))(
    ins:{
        duration; opora;
        files: Range(1,0)
    },
    outs:{
        corr_output: Range(1,0)
    })
```

Рисунок 3.7 – Внутреннее представление плана для общей задачи вычисления сейсмической свёртки

При дальнейшем планировании на частной задаче было получено ожидаемое множество из 100 вхождений каждой из операций. Результат представлен на рисунке 3.8. Зелёным цветом выделены множества определения операций.

Данный пример показывает способность алгоритма выделять диапазоны однотипных операций. Причём диапазоны выделяются автоматически, что подтверждает соответствие требованиям к алгоритмической сложности.

```txt
{
    'opora': opora(entry:['corr_cpu', 'corr_gpu'], comp:[]),
    'duration': duration(entry:['corr_cpu', 'corr_gpu'], comp:[]),
    'corr_output': corr_output(entry:[], comp:['corr_cpu', 'corr_gpu']),
    'files': files(entry:['corr_cpu', 'corr_gpu'], comp:[])
}
corr_cpu: corr_cpu[{
    corr_cpu: {0,1,...,99}(step=1)
}](
    ins:[('files', (1, 0)), 'opora', 'duration'],
    outs:[('corr_output', (1, 0))]
  )

corr_gpu: corr_gpu[{
    corr_gpu: {0,1,...,99}(step=1)
}](
    ins:[('files', (1, 0)), 'opora', 'duration'],
    outs:[('corr_output', (1, 0))]
  )
```

Рисунок 3.8 – Внутреннее представление плана для частной задачи вычисления сейсмической свёртки

3.3.3 Задача моделирования потока жидкости

В работе [25] рассматривается клеточно-автоматная модель потока жидкости. Предметная область задачи на высоком уровне может быть представлена в виде множества состояний клеточного автомата и двух функций перехода – сдвиг и столкновение. Здесь для удобства тестирования будет введена дополнительная операция визуализации состояния автомата. На рисунке 3.9 представлена возможная ВМ для данной задачи.

В таком варианте используются линейные зависимости с коэффициентом два. Можно было также выделить промежуточное состояние в отдельный массив. Также важным преимуществом подхода оказалась гибкость при выборе смещения в линейных зависимостях. Так, при описании операции collision использовалось начальное смещение не для второго и третьего состояний, а для нулевого и первого (как обобщение чётных и нечётных состояний), а при планировании автоматически выделилось ограничение $i \geq 1$.

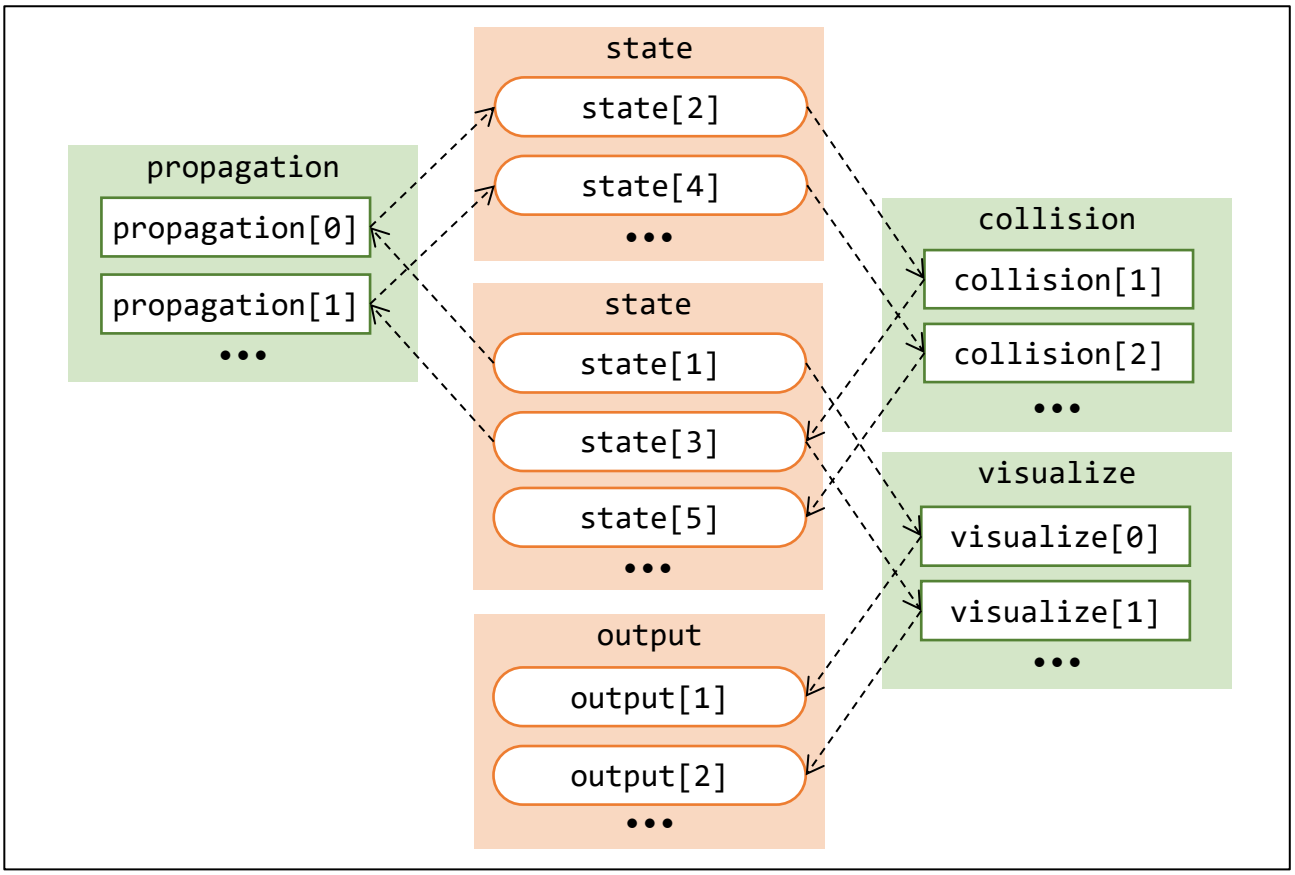

Рисунок 3.9 – Вычислительная модель клеточного-автоматной модели FHP-MP

Общая задача на ВМ формулируется так: задано начальное состояние модели (первая компонента массива); требуется найти все состояния и все визуализации состояний. В частной задаче на ВМ указывается, например, конкретное изображение. При тестировании строился план вычисления изображения 20000-ого состояния. Ожидалось автоматическое попадание в план всех операций, вычисляющих промежуточные состояния автомата.

Программа, выполняющая планирование для такой задачи, с использованием библиотеки может быть записана так, как показано на рисунке 3.10.

При запуске системы планирования были сконструированы две пси-операции. Последовательная задаёт вычисление из первого состояния всех остальных, а автоматическая задаёт вычисление изображение для произвольного состояния. Результат (внутренне представление промежуточной ВМ) представлен на рисунке 3.11. Зелёным цветом выделены множества термов, описываемых пси-операциями.

Видно, что два вхождения операции *visualize* не попали в пси-операцию, поскольку на начальных этапах восходящей части ещё не была выявлена последовательная пси-операция. Подобные случаи не вызывают трудностей при дальнейшем планировании, поскольку таких вхождений конечное и относительно (в масштабах задачи) небольшое число.

```python
# from {...} import *

# псевдонимы для сокращения записей
R = Range
VS = VariablesSet

modelOperations=[
    MOperation('propagation',
               ins=[('state',(2,1))],
               outs=[('state',(2,2))],
               scope=VS([('propagation',[R((1,0))])])
               ),
    MOperation('collision',
               ins=[('state',(2,0))],
               outs=[('state',(2,1))],
               scope=VS([('collision',[R((1,0))])])
               ),
    MOperation('visualize',
               ins=[('state',(2,1))],
               outs=[('output',(1,1))],
               scope=VS([('visualize',[R((1,0))])])
               )
]
modelVariables=[
    MVariable('state',
              entry=['propagation', 'collision', 'visualize'],
              comp=['propagation', 'collision']
              ),
    MVariable('output',
              entry=[], comp=['visualize']
              )
]
fhpmp_model = Model(v={x.label : x for x in modelVariables},
              o={x.label : x for x in modelOperations})

# планирование на общей задаче
fhpmp_model_general = model_plan(fhpmp_model,
                                 V=VS([('state',[R(1)])]),
                                 W=VS([('output',[R((1,1))])])
                                 )
# планирование на частной задаче
fhpmp_model_specific = model_fit(fhpmp_model, fhpmp_model_general,
                                 V=VS([('state',[R(1)])]),
                                 W=VS([('output',[R(20000)])])
                                 )
```

Рисунок 3.10 – Использование библиотеки при планировании для задачи на клеточного-автоматной модели FHP-MP

```txt
{
    'state': state(entry:['visualize', 'psi0', 'psi1'], comp:['psi0']),
    'output': output(entry:[], comp:['visualize', 'psi1'])
}
visualize: visualize[{
    visualize: {0,1}
}](
    ins:[('state', (2, 1))],
    outs:[('output', (1, 1))]
  )

psi0: psi0(psi: collision[(1, 1)](propagation[(1, 0)](state[(2, 1)])))(
    ins:{
        state:  {1}
    },
    outs:{
        state: Range(1,2)
    })

psi1: psi1(psi(auto): visualize[(1, 2)](state[(2, 5)]))(
    ins:{
        state: Range(2,5)
    },
    outs:{
        output: Range(1,3)
    })
```

Рисунок 3.11 – Внутреннее представление плана для общей задачи моделирования потока жидкости

При дальнейшем планировании на частной задаче было получено, что для вычисления **W** необходимо выполнить по 19999 вхождений операций сдвига и столкновения, а также только 19999-ое вхождение визуализации. Результат представлен на рисунке 3.12. Зелёным цветом выделены множества определения операций.

В данном случае операции нельзя выполнять параллельно, но результат показывает весь потенциал алгоритма планирования с последовательными пси-операциям. Для определения последовательности из чередующихся вхождений операций потребовалось достичь лишь первых двух первых вхождений, а остальные были вычислены из найденной зависимости.

```txt
{
    'state': state(
        entry:['propagation', 'collision', 'visualize'],
        comp:['propagation', 'collision']
        ),
    'output': output(entry:[], comp:['visualize'])
}
propagation: propagation[{
    propagation: {0,1,...,19998}(step=1)
}](
    ins:[('state', (2, 1))],
    outs:[('state', (2, 2))]
  )

collision: collision[{
    collision: {1,2,...,19999}(step=1)
}](
    ins:[('state', (2, 0))],
    outs:[('state', (2, 1))]
  )

visualize: visualize[{
    visualize: {19999}
}](
    ins:[('state', (2, 1))],
    outs:[('output', (1, 1))]
  )
```

Рисунок 3.12 – Внутреннее представление плана для частной задачи моделирования потока жидкости

3.3.4 Результаты тестирования

В ходе тестирования были составлены вычислительные модели двух задач численного моделирования. Также были сформулированы задачи на вычислительных моделях. Представление получилось достаточно простым и интуитивно понятным. Не возникает сомнений, что подобный подход можно использовать и для многих других задач с похожей структурой.

Проблем с описанием общей и частной задач не возникло. Учитывая преимущества такого подхода, это невысокая цена за возможность задания ограничений вычислений, которые иначе могут быть достигнуты при использовании более сложных вычислительных моделей с предикатами и счётчиками [5].

Также желаемые результаты были получены для ряда простых задач, включая пример из описания алгоритма имитации планирования и вычисление чисел Фибоначчи. В первом случае структура задачи совпадает с ВМ для клеточно-автоматной модели FHP-MP. Описание тестирования для второй задачи приведено в приложении Г.

ЗАКЛЮЧЕНИЕ

В ходе работы были исследованы практические возможности реализации системы планирования на вычислительных моделях с массивами.

Основными теоретическими результатами работы являются: идея планирования в несколько этапов (а именно, на частной и общей задачах), а также интерпретация алгоритма планирования [7] в терминах компьютерных программ.

В ходе работы реализована библиотека планирования для ВММ и форматы представления вычислительных моделей в терминах объектно-ориентированного программирования и в универсальном текстовом описании. Также частично проведена интеграция с системой активных знаний, для чего был реализован серверный компонент для библиотеки планирования.

В работе рассмотрены два основных класса задач, которые могут решаться при помощи ВММ. В том числе продемонстрирована возможность использования предлагаемого подхода для решения практически значимых задач численного моделирования.

Используемый подход имеет ряд ограничений и недостатков, часть из которых была решена в данной работе, а также были определены важные этапы дальнейшего развития в области синтеза параллельных программ на ВММ. Многие моменты, связанные с реализацией алгоритмов и описанные в работе, могут быть использованы для разработки более универсальных и эффективных подходов.

Основные защищаемые положения:

а) Разработка модификации алгоритма планирования на ВММ и её детализированное описание.

б) Разработка подхода к представлению и формата описания ВММ и задач на ВММ.

в) Программная реализация модуля планирования, состоящего из библиотеки планирования и серверного компонента.

Далее планируется разработать подход к представлению многомерных множеств объектов ВММ и расширить алгоритм планирования для работы с ними. Также планируется расширение класса поддерживаемых задач – для этого требуется использовать другие способы задания ограничений вычислений (помимо разделения частной и общей формулировок задач).

Выпускная квалификационная работа выполнена мной самостоятельно и с соблюдением правил профессиональной этики. Все использованные в работе материалы и заимствованные принципиальные положения (концепции) из опубликованной научной литературы и других источников имеют ссылки на них. Я несу ответственность за приведенные данные и сделанные выводы.

Я ознакомлен с программой государственной итоговой аттестации, согласно которой обнаружение плагиата, фальсификации данных и ложного цитирования является основанием для не допуска к защите выпускной квалификационной работы и выставления оценки «неудовлетворительно».

_____ _____

ФИО студента *Подпись студента*

«____» _____20 __г.

(заполняется от руки)

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Крюков В. А. Разработка параллельных программ для вычислительных кластеров и сетей // Информационные технологии и вычислительные системы. – 2003. – № 1–2. – С. 42–61.
2. MPI Documents. Официальная документация стандарта MPI [Электронный ресурс]. URL: https://www.mpi-forum.org/docs/ (дата обращения 27.04.2023).
3. Антонов А. С. Параллельное программирование с использованием технологии OpenMP / Москва : Издательство Московского университета. – 2009. – 76 с.
4. Бондаренко В.В., Козич В.Г., Плахотная Л.А. Достоинства и недостатки параллельного программирования // Современные научные исследования и инновации. – 2016. – № 6 [Электронный ресурс]. URL: https://web.snauka.ru/issues/2016/06/69538 (дата обращения: 28.04.2024).
5. Вальковский В. А., Малышкин В. Э. Синтез параллельных программ и систем на вычислительных моделях / Новосибирск : Наука, 1988. – 129 с.
6. Malyshkin V. Active knowledge, LuNA and literacy for oncoming centuries. In: Bodei C, Ferrari G, Priami C (eds) Programming languages with applications to biology and security // Lecture notes in computer science. – Springer, Cham. – 2015. – Volume 9465. – doi: 10.1007/978-3-319-25527-9.
7. Малышкин В. Э., Панкратов С. А. Алгоритм имитации планирования // Вычислительные системы. – 1986. – № 114. – С. 122-136.
8. Тыугу Э. Х. Решение задач на вычислительных моделях // Журнал вычислительной математики и математической физики. – 1970. – Т. 10, № 3. – С. 716–733.
9. Кахро М. И., Калья А. П., Тыугу Э. Х. Инструментальная система программирования ЕС ЭВМ (ПРИЗ) / Москва : Финансы и статистика. – 1981. – 158 с.

10. Карзанов А. В., Фараджев И. А. Планирование вычислений при решении задач на вычислительных моделях // Программирование. – 1975. – № 4. – С. 18-24.
11. Парфенов Д. Р., Мокроусов А. С. Алгоритм планирования на вычислительных моделях с учётом нефункциональных требований на основе моделирования исполнения // Параллельные вычислительные технологии ПаВТ’2024. – Челябинск : Издательский центр ЮУрГУ. – 2024. – С. 190. – doi: https://doi.org/10.14529/pct2024.
12. Перепёлкин В. А. Язык и система LuNA для автоматического конструирования программ численного моделирования // Проблемы информатики. – 2020. – № 1. – С. 66–74.
13. Новосельцев В. Б. Теория структурных функциональных моделей // Сибирский математический журнал. – 2006. – Т. 47, № 6. – С. 1342–1354.
14. Novoseltsev V. B. Synthesis of parallel recursive programs in structural functional models // Programming and Computer Software. – 2007. – Volume 33, Issue No 5. – P. 293–298. – doi:10.1134/s0361768807050052.
15. Патрикеев Е. М. Автоматический синтез параллельных программ // Наукоёмкие информационные технологии. – 2010. – С. 133–142.
16. HPC Community Cloud | Supercomputer Software Department. Описание проекта HPC Community Cloud [Электронный ресурс]. URL: https://ssd.sscc.ru/ru/projects/hpc-community-cloud (дата обращения: 28.04.2024).
17. Gorodnichev M., Lebedev D. Semantic tools for development of high-level interactive applications for supercomputers. // The Journal of Supercomputing. – 2021. – doi:10.1007/s11227-021-03731-6.
18. Public Key Encription and Number Theory. Описание понятий и программных реализаций алгоритмов теории чисел [Электронный ресурс]. URL: http://mech.math.msu.su/~vvb/Master/NumberTheory.html (дата обращения: 19.05.2024).

19. Yogesh Rana. Python: Simple though an Important Programming language // International Research Journal of Engineering and Technology. – 2019. – Volume 6, Issue No 2. – P. 1856–1858.
20. Малышкин В. Э., Перепёлкин В. А. Построение баз активных знаний для автоматического конструирования решений прикладных задач на основе системы LuNA // Параллельные вычислительные технологии ПаВТ’2024. – Челябинск : Издательский центр ЮУрГУ. – 2024. – С. 126-136. – doi: https://doi.org/10.14529/pct2024.
21. Tornado Web Server – Tornado 6.4 documentation. Официальная документация фреймворка Tornado [Электронный ресурс]. URL: https://www.tornadoweb.org/en/stable/ (дата обращения: 29.04.2024).
22. Karavaev D. A., Yakimenko A. A., Bulavina N. A. A technology of full seismic field simulation on high-performance computing systems // 2016 13th International Scientific-Technical Conference on Actual Problems of Electronics Instrument Engineering (APEIE). – 2016. – doi:10.1109/apeie.2016.7806368.
23. Vyrodov A.V., Perepelkin V. A., Khyretdinov M. S., Khrypchenko A. V. Principles of organizing a software-analytical system for parallel processing of seismic data. // 2023 IEEE XVI International Scientific and Technical Conference APEIE. – 2023.
24. Хайретдинов М. С., Воскобойникова Г. М., Седухина Г. Ф. Алгоритмы поточной свертки в задачах активного вибросейсмоакустического мониторинга // Интерэкспо Гео-Сибирь. – Новосибирск. – 2017. – С. 230–234.
25. Медведев Ю. Г. Многочастичная клеточно-автоматная модель потока жидкости FHP-MP // Вестник Томского государственного университета. Управление, вычислительная техника и информатика. – 2009. – № 1(6). – С. 33-40.

ПРИЛОЖЕНИЕ А

Алгоритм сокращения множества элементов массива

Описание и общие сведения:

Данный алгоритм применяется при вставке любого элемента в Множество и приводит множество в стандартный вид согласно приоритетам для последовательностей. На этапе работы гарантируется, что Множество находится в сокращённом виде, за исключением, возможно, вставленной на текущем шаге последовательности. Также гарантируется, что вставленная последовательность не имеет общих элементов ни с какой другой последовательностью Множества.

Для каждой последовательности r определены числа $r.a$, $r.b$, $r.c$, $r.e$ – шаг, начальный элемент, количество элементов (0 для неограниченной) и последний элемент (имеет смысл только для ограниченных) соответственно.

Если последовательность ограниченная, то шаг всегда положительное число. Если последовательность содержит единственный элемент, то шаг считается равным 1.

Входные данные: вставленная на текущем шаге последовательность r и множество M всех последовательностей Множества.

Выходные данные: отсутствуют. Изменяется только само Множество при помощи стандартных методов вставки и удаления последовательностей.

Общий алгоритм:

1) Если r не ограничена:

1.1) Найти p – предыдущий элемент последовательности r ($r.b - r.a$).

1.2) Для каждой последовательности i ($i \neq r$) в множестве M:

1.2.1) Если i неограниченная, перейти к следующие последовательности в (1.2).

1.2.2) Найти пересечение s последовательности i и неограниченной последовательности с шагом $-r.a$ и начальным элементом $r.b$.

1.2.3) Если p не содержится в s, перейти к следующей последовательности в (1.2).

1.2.4) Если $s.a > |r.a|$, заменить s на диапазон из одного элемента p.

1.2.5) Удалить i и r из M.

1.2.6) Добавить в M последовательность с шагом $r.a$ и началом в $s.b$ (при $r.a > 0$), или в $s.e$ (иначе).

1.2.7) Добавить в M каждую последовательность из $i - s$.

1.2.8) Завершить алгоритм.

1.3) Если $r.a = 1$, завершить алгоритм.

1.4) Создать пустой словарь C.

1.5) Для каждой последовательности i ($i \neq r$) в множестве M:

1.6) Если i ограниченная и $i.a = r.a$ (пусть $a \coloneqq r.a$):

1.6.1) Вычислить $s \coloneqq ((r.b - i.b)\ mod\ a + a)\ mod\ a$

1.6.2) В словаре C увеличить значение под ключом $GCD(s, a)$ на 1, (изначально равно 1)

1.7) Для каждой пары ключ-значение (k, v) в C в порядке возрастания k:

1.7.1) Если $v \cdot k = a$:

1.7.1.1) Определить множество $\mathrm{M}' = \{\mathrm{i} : (\mathrm{i} \in M) \wedge (i.a = r.a) \wedge ((r.b - i.b) = 0\ mod\ k)\}$.

1.7.1.2) Найти b $\coloneqq$ $\max(\{i.b : i \in M'\})$ при $a > 0$, или $\min(\{i.b : i \in M'\})$ при $a < 0$.

1.7.1.3) Для всех $i \in M'$:

1.7.1.3.1) Удалить i из M.

1.7.1.3.2) Если $i.b < b$ ($i.b > b$ для $a < 0$), добавить в M последовательность от $i.b$ (включительно) до b (не включительно) с шагом a.

1.7.1.4) Добавить неограниченную последовательность с шагом k и началом в b.

2) Иначе (r ограничена):

2.1) Для каждой последовательности i ($i \neq r$) в множестве M:

2.1.1) Если i ограниченная, перейти к следующие последовательности в (2.1).

2.1.2) Найти p – предыдущий элемент последовательности i ($i.b - i.a$).

2.1.3) Найти пересечение s последовательности r и неограниченной последовательности с шагом $-i.a$ и начальным элементом $i.b$.

2.1.4) Если p не содержится в s, перейти к следующей последовательности в (2.1).

2.1.5) Если $\mathrm{s.a} > |i.a|$, заменить s на диапазон из одного элемента p.

2.1.6) Удалить i и r из M.

2.1.7) Добавить в M последовательность с шагом $i.a$ и началом в $s.b$ (при $i.a > 0$), или в $s.e$ (иначе).

2.1.8) Добавить в M каждую последовательность из $r - s$.

2.1.9) Завершить алгоритм.

2.2) Для каждой последовательности i ($i \neq r$) в множестве M:

2.2.1) Если i неограниченная, перейти к следующие последовательности в (2.2).

2.2.2) Выполнить *алгоритм сокращения для двух ограниченных последовательностей*.

Алгоритм сокращения для двух ограниченных последовательностей:

Алгоритм выполняется в общем алгоритме сокращения для двух ограниченных последовательностей r и i.

1) Если $r.c = 1$:

1.1) Если $r.b < i.b$:

1.1.1) Если $i.b - r.b < i.a$ или $i.c = 1$:

1.1.1.1) Удалить из M последовательности r и i.

1.1.1.2) Добавить в M последовательность из двух элементов – $r.b$ и $i.b$.

1.1.1.3) Если $i.c > 1$, добавить в M последовательность от $i.b + i.a$ (включительно) до $i.e$ (включительно) с шагом $i.a$.

1.1.1.4) Завершить алгоритм.

1.1.2) Если $i.b - r.b = i.a$

1.1.2.1) Удалить из M последовательности r и i.

1.1.2.2) Добавить в M последовательность от $r.b$ (включительно) до $i.e$ (включительно) с шагом $i.a$.

1.1.2.3) Завершить алгоритм.

1.2) Если $r.b > i.e$:

1.2.1) Аналогично (1.1), только единственный элемент r объединяется либо с последним элементом i, либо добавляется в конец последовательности i.

1.3) Иначе (элемент r находится между границами последовательности i):

1.3.1) Если $(r.b - i.b) \bmod \mathrm{i.a} < (i.e - r.b) \bmod \mathrm{i.a}$

1.3.1.1) Из M удаляются последовательности r и i.

1.3.1.2) $d \coloneqq (r.b - i.b)\ mod\ i.a$.

1.3.1.3) Добавить в M двухэлементный диапазон с $r.b$ и $r.b - d$.

1.3.1.4) Если $r.b - d > i.b$, добавить в M диапазон с $i.b$ (включительно) до $r.b - 2 \cdot d$ (включительно) с шагом $i.a$.

1.3.1.5) Если $r.b < i.e$, добавить в M диапазон с $r.b - i.a + d$ (включительно) до $i.e$ (включительно) с шагом $i.a$.

1.3.1.6) Завершить алгоритм.

1.3.2) Иначе:

1.3.2.1) Аналогично (1.3.1), только единственный элемент r объединяется с правым ближайшим соседним элементом из i.

2) Иначе ($r.c > 1$):

2.1) Если $i.c = 1$:

2.1.1) Аналогично (1.1) – (1.2).

2.2) Если $r.a = i.a$:

2.2.1) Если $r.e + r.a = i.b$ или $i.e + i.a = r.b$:

2.2.1.1) Удалить последовательности r и i из M.

2.2.1.2) Добавить в М последовательность, содержащую обе последовательности с тем же шагом.

2.2.1.3) Завершить алгоритм

2.3) Если $r.e > i.b$ и $r.b < i.e$ (возможно пересечение):

2.3.1) Удалить последовательности r и i из M.

2.3.2) Последовательности упорядочиваются по начальному (левому) элементу. Далее $\{r_1, r_2\} = \{r, i\}, r1.b > r2.b$.

2.3.3) Вычисляется $nok = \text{НОК}(r_1.a, r_2.a)$.

2.3.4) Вычисляется $s = (\mathrm{r}_2.\mathrm{e} - \mathrm{r}_1.\mathrm{b}) \bmod \mathrm{r}_2.\mathrm{a}$.

2.3.5) Создаётся и сортируется по возрастанию список S из элементов двух списков: $[0, r_1.a, 2 \cdot r_1.a, \dots, \min(nok, r_1.e - r_1.b + 1))$ и $[s,\ s + r_2.a, s + 2 \cdot r_2.a, \dots, \min(nok, r_2.e - r_2.b + 1))$.

2.3.6) Далее происходит обход списка S и в нём выделяются цепочки равноудалённых элементов. Если очередной элемент находится на большем расстоянии, то формируется цепь, а текущий элемент образует новую (незавершённую). Если очередной элемент находится на меньшем расстоянии, то последний элемент и текущий образуют новую цепь (незавершённую), а остальные элементы образуют готовую цепь. При совпадении расстояний, элемент добавляется к текущей цепи. В конце обхода цепь принудительно завершается.

2.3.7) Если образована одна цепь, то в M добавляется последовательность от $r_1.b$ до $min(r_1.e, r_2.e)$ с шагом этой цепи. Оставшиеся части последовательностей после $min(r_1.e, r_2.e)$ добавляются с исходным шагом.

2.3.8) Если образовано несколько последовательностей:

2.3.8.1) Каждая цепь добавляется в виде отдельной последовательности для каждого возможного смещения от $r_1.b$ до $min(r_1.e, r_2.e)$ с шагом nok.

2.3.9) Добавить в M часть последовательности r_2 от $r_2.b$ до $r_1.b$.

ПРИЛОЖЕНИЕ Б

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

МОДУЛЬ ПЛАНИРОВАНИЯ НА ВЫЧИСЛИТЕЛЬНЫХ МОДЕЛЯХ

«CM4LuNA»

Описание программы

Листов 10

2024

Литера «Т»

АННОТАЦИЯ

В данном документе приведено описание библиотеки препланирования на вычислительных моделях с массивами. Библиотека для языка Python предоставляет классы для описания вычислительных моделей и задач на них, а также процедуры планирования их вычисления.

В документе описаны форматы, принятые в библиотеке, а также описан алгоритм работы процедур планирования.

Оформление программного документа «Описание программы» произведено по требованиям ГОСТ 19.402-78 «ЕСПД. Описание программы» и ГОСТ 19.105-78 «Единая система программной документации (ЕСПД). Общие требования к программным документам (с Изменением № 1)».

СОДЕРЖАНИЕ

1. Общие сведения 65
2. Функциональное назначение 65
3. Описание логической структуры 65
   3.1. Алгоритм программы и используемые методы 65
   3.2. Структура программы 66
4. Используемые технические средства 67
5. Вызов и загрузка 67
6. Входные данные 67
   6.1. Общее описание 67
   6.2. Описание классов и процедур библиотеки 68
   6.3. Описание JSON формата 70
7. Выходные данные 71

1. ОБЩИЕ СВЕДЕНИЯ

Полное название программы: модуль планирования на вычислительных моделях «CM4LuNA».

Краткое название программы: «CM4LuNA».

Программа требует установленного интерпретатора языка Python 3 версии.

Программа написана на языке Python.

2. ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ

Функциональным назначением модуля «CM4LuNA» является построение плана вычислений на вычислительной модели. Класс решаемых задач включает вычислительные модели с массивами и простые вычислительные модели. Ограничения на используемые вычислительные модели с массивами:

1) Использование только одномерных массивов.

2) Использование линейных функций зависимостей в массовых операциях.

Полный класс решаемых задач не исследовался.

Эксплуатационным назначением модуля «CM4LuNA» является использование при конструировании параллельной программы.

3. ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ

3.1. Алгоритм программы и используемые методы

Планирование проводится в два этапа – на общей и частной задачах. При планировании на общей задаче выполняется восходящая часть планирования – обход объектов вычислительной модели в ширину от заданного множества V задачи, во время которого выделяются пси-операции. Такие операции задают бесконечные множества вхождений операций, и вычислительная модель с массивами становится аналогичной простой вычислительной модели. Далее выполняется нисходящая часть – обход полученного множества объектов от множества W задачи. В конце этапа определяется достижимость W по V.

При планировании на частной задаче сначала выполняется нисходящая часть, начиная с W частной задачи. На этом этапе раскручиваются пси-операции до реально используемых диапазонов вхождений. Далее выполняется восходящая часть для конечного подмножества исходной вычислительной модели, полученного в ходе предыдущих шагов. На этом этапе выделяются все необходимые и достаточные операции, что и является планом для конкретной задачи на вычислительной модели.

3.2. Структура программы

Библиотека разделена на модули (файлы), представленные в таблице Б.1.

Таблица Б.1 – список модулей библиотеки планирования

Файл	**Размер, строк**	**Описание**
alglogger.py	34	Работа с протоколированием в ходе работы.
base_math.py	32	Вычисление НОК, НОД. Вариант расширенного алгоритма Евклида.
fit_tools.py	231	Вспомогательные процедуры при планировании на частной задаче.
json_fields_names.py	31	Описание полей JSON-формата ВМ.
model.py	238	Описание классов объектов ВМ.
plan.py	238	Основные (высокоуровневые) процедуры планирования.
plan_tools.py	287	Вспомогательные процедуры при планировании на общей задаче.
ranges.py	246	Класс диапазонов.
terms.py	82	Работа с термами при построении пси-операций.
variables_set.py	638	Классы множеств для хранения диапазонов.

Зависимости между модулями библиотеки, а также взаимодействие с пользователем представлены на рисунке Б.1.

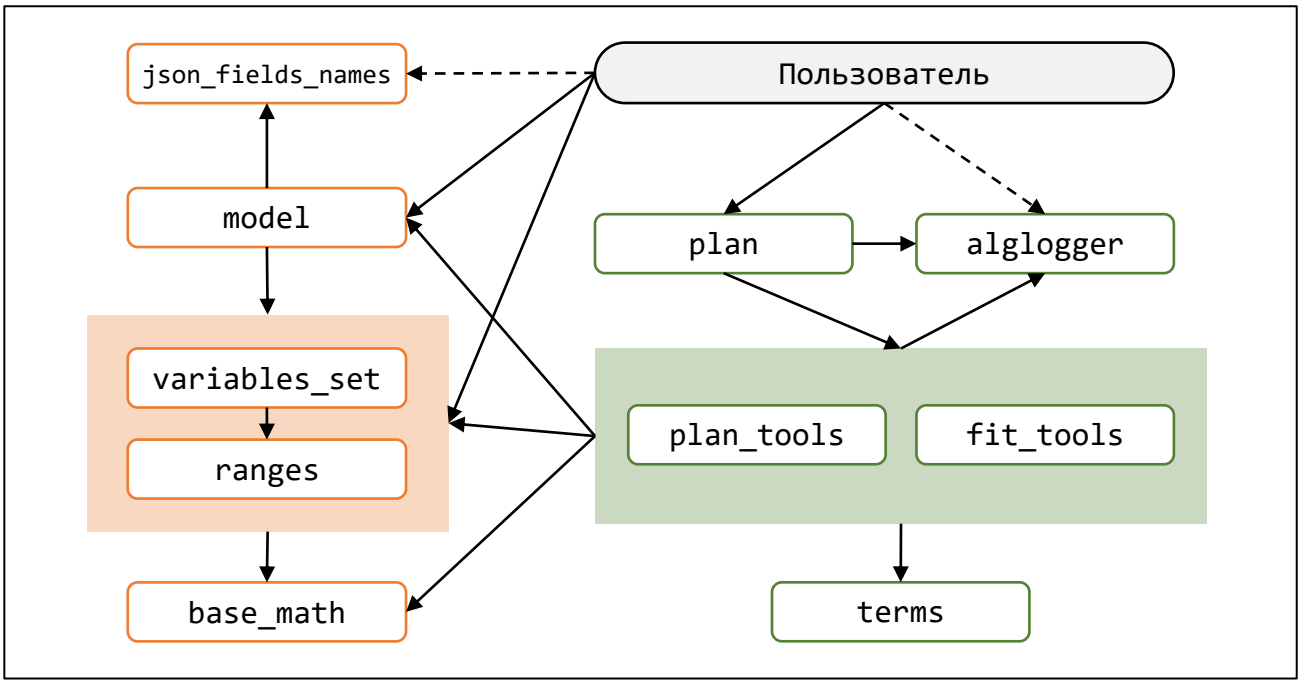

Рисунок Б.1 – Схема взаимодействия модулей библиотеки планирования

4. ИСПОЛЬЗУЕМЫЕ ТЕХНИЧЕСКИЕ СРЕДСТВА

Программа может исполняться на персональных компьютерах, поддерживаемых интерпретатором языка Python. Дополнительных требований к техническим средствам не предъявляется, поскольку объём используемых при работе ресурсов не исследовался для сложных задач.

5. ВЫЗОВ И ЗАГРУЗКА

Программа используется как библиотека для языка Python. Входной точкой является вызов функций планирования для созданных в программе внутренних представлений вычислительной модели и задач.

6. ВХОДНЫЕ ДАННЫЕ

6.1. Общее описание

Входными данными являются объекты в системе Python, реализующие классы внутреннего представления алгоритма планирования. Входные данные

передаются через именованные аргументы в процедуры планирования. Способ загрузки определяется пользователем. Предусмотрен способ создания объекта вычислительной модели из строкового представления JSON-формата.

6.2. Описание классов и процедур библиотеки

Диапазон (класс Range).

Атрибуты диапазона представлены объектами класса. Причём одинаковое представление используется как для ограниченных, так и для неограниченных диапазонов. Ограниченность определяется по параметру «количество элементов» – для не ограниченных он равен нулю. Для удобства методы пересечения и вычитания двух диапазонов переопределены для операторов «умножить» и «минус» соответственно. При этом оператор «минус» является итератором по всем диапазонам результирующего множества.

Множество (класс VariablesSet).

Конструктор принимает список элементов множества. Каждый элемент является именем, либо кортежем из имени и списка из одного диапазона.

Поддерживает добавление и удаление элементов. Реализованы изменяющие и копирующие операции объединения, пересечения, вычитания множеств. Также реализован итератор по множеству. Пример такого объекта с комментариями представлен на рисунке Б.2.

Операции.

Представлены двумя классами для простых (*SOperation*) и массовых (*MOperation*) операций. Содержат два списка входных (ins) и выходных (outs) переменных. Списки содержат имена простых переменных, либо кортежи с именем массива и кортежем (a, b) линейной функции использования. Для массовых операций указывается множество применимости *scope* типа *VariablesSet*. Множество применимости содержит единственное имя – саму операцию.

Переменные.

Представлены двумя классами для простых (*SVariable*) и массовых (*MVariable*) переменных. Классы содержат два списка использующих

переменную операций – как входную (*entry*) и как выходную (*comp*). Операции в них всегда представлены только именем операции.

Вычислительная модель (класс Model).

Содержит два словаря для переменных (*v*) и операций (*o*). Ключами являются имена, а значениями – объекты описанных выше классов.

Функция планирования на общей задаче (model_plan).

Принимает объект Model и два объекта *VariablesSet* в именованных параметрах *V* и *W* для соответствующих множеств задачи. Возвращается копия исходной вычислительной модели с новыми пси-операциями, а также с отфильтрованными в ходе работы множествами переменных и операций.

Функция планирования на частной задаче (model_fit).

Принимает два объекта *Model* для исходной вычислительной модели и результата работы *model_plan*, и два объекта *VariablesSet* в именованных параметрах *V* и *W* для соответствующих множеств задачи. Возвращается копия исходной вычислительной модели с отфильтрованными в ходе работы множествами переменных и операций.

python

```python
V= VariablesSet(
  [
    ('x', [Range(1)]),
    # x[1]

    ('z', [Range(1,10,2)]),
    # z[1], z[3], z[5], z[7], z[9]

    ('y', [Range((2,0))]),
    # y[0], y[2], y[4], ...

    ('y', [Range((3,0))])
    # y[0], y[3], y[6], ...
  ]
).difference( # разность с другим множеством
  VariablesSet([
    ('y', [Range((6,10))]
    # y[10], y[16], y[22], ...
  ])
)
```

txt

```txt
Реальный объект V (после
сокращения представления):

{
x: {1};
z: {1,3,...,9}(step=2);
y: Range(3,0),
   {4},
   Range(6,2)
}


Расшифровка диапазонов:

x[1]
z[1], z[3], z[5], z[7],
z[9]
y[0], y[3], y[6], ...
y[2], z[8], z[14], ...
y[4]
```

Рисунок Б.2 – Пример описания объекта класса VariablesSet

6.3. Описание JSON формата

JSON-объект вычислительной модели содержит два объекта – словари переменных (*variables*) и операций (*operations*), – а также поля с пользовательской информацией – имя (*name*) или описание (*description*). Ключами являются id переменных и операций, а значениями – объекты со свойствами операции или переменной. Для каждого объекта необходимо поле *type* с возможными значениями *simple* и *massive* для простых объектов и массовых соответственно. Также для каждого объекта допустимы поля с пользовательской информацией – имя (*name*), описание (*description*).

Для объекта массовой операции необходимо поле *scope*, которое задаёт область определения операции. *scope* представляет массив 3-х элементных массивов вида $[a, b, c]$, каждый из которых задаёт последовательность с шагом a, начиная с b. Элемент c здесь задаёт количество элементов, и, если c равно нулю, последовательность считается неограниченной. Например, объект на рисунке Б.3 задаёт множество $\{0,1,2,\ldots,48,49,51,53,\ldots,67,69,70,73,\ldots\}$.

```
json
"scope": [
  [1, 0, 50],
  [2, 51, 10],
  [3, 70, 0]
]
```

Рисунок Б.3 – Пример JSON описания множества диапазонов

Отношения между переменными и операциями задаётся в объектах операций – поля *inputs* и *outputs* для входных и выходных использований соответственно. Эти поля задают массив, в котором каждый объект содержит либо только поле *id* (для простых переменных), либо поля *id* и *scope* (для массивов). *scope* является массивом вида $[a, b]$, что означает использование в i-ом вхождении операции $a \cdot i + b$ вхождения массива. При использовании элементов массивов в простых операциях, значение a должно быть равно нулю.

На рисунке Б.4 приведён пример описания в предложенном формате вычислительной модели с двумя массивами x и y, простой переменной n и одной массовой операцией f, вычисляющей $y[i]$ по $x[i]$ и n для натуральных i.

```json
{
  "name": "пример ВМ",
  "variables": [
    { "id": "n", "type": "simple"  },
    { "id": "x", "type": "massive" },
    { "id": "y", "type": "massive" }
  ],
  "operations": {
    "f": {
      "type": "massive",
      "description": "вычисляет (x[i], n) -> y[i]",
      "inputs": [
        { "id": "n" },
        { "id": "x", "scope": [1, 0] }
      ],
      "outputs": [
        { "id": "y", "scope": [1, 0] }
      ],
      "scope": [[1, 1, 0]]
    }
  }
}
```

Рисунок Б.4 – Пример JSON описания вычислительной модели

7. ВЫХОДНЫЕ ДАННЫЕ

Входными данными являются объекты в системе Python, реализующие классы внутреннего представления алгоритма планирования. Формат совпадает с форматом входных данных. Результат планирования возвращается из функции в основную программу. Дальнейшая обработка результата определяется пользователем.

В случае невозможности выполнить планирование из-за неверно поставленной задачи, в качестве результата возвращается None, то есть нулевое значение языка Python.

Лист регистрации изменений									
Номера листов (страниц)					Всего листов (страниц) в докум.	№ документа	Входящий № сопроводительного докум. и дата	Подп.	Дата
№ изм.	измененных	замененных	новых	аннулированных					

ПРИЛОЖЕНИЕ В

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

МОДУЛЬ ПЛАНИРОВАНИЯ НА ВЫЧИСЛИТЕЛЬНЫХ МОДЕЛЯХ

«CM4LuNA»

Руководство программиста

Листов 5

2024

Литера «Т»

АННОТАЦИЯ

В данном документе приведено руководство программиста для библиотеки планирования на вычислительных моделях с массивами .

Библиотека для языка Python предоставляет классы для описания вычислительных моделей и задач на них, а также процедуры планирования их вычисления.

В документе описаны назначение программы, способы запуска и использования программы, а также действия, выполняемые при возникновении ошибок.

Оформление программного документа «Руководство программиста» произведено по требованиям ГОСТ 19.504-79 «ЕСПД. Руководство программиста» и ГОСТ 19.105-78 «Единая система программной документации (ЕСПД). Общие требования к программным документам (с Изменением № 1)».

СОДЕРЖАНИЕ

1. Назначение и условия применения программ 76
2. Характеристика программы 76
3. Обращение к программе 77
4. Входные и выходные данные 77
5. Сообщения 77

1. НАЗНАЧЕНИЕ И УСЛОВИЯ ПРИМЕНЕНИЯ ПРОГРАММ

Полное название программы: модуль планирования на вычислительных моделях «CM4LuNA».

Краткое название программы: «CM4LuNA».

Программа требует установленного интерпретатора языка Python 3 версии.

Функциональным назначением модуля «CM4LuNA» является построение плана вычислений на вычислительной модели. Класс решаемых задач включает вычислительные модели с массивами и простые вычислительные модели. Ограничения на используемые вычислительные модели с массивами:

1) Использование только одномерных массивов.

2) Использование линейных функций зависимостей в массовых операциях.

Полный класс решаемых задач не исследовался.

Эксплуатационным назначением модуля «CM4LuNA» является использование при конструировании параллельной программы.

Программа может исполняться на персональных компьютерах, поддерживаемых интерпретатором языка Python. Дополнительных требований к техническим средствам не предъявляется, поскольку объём используемых при работе ресурсов не исследовался для сложных задач.

2. ХАРАКТЕРИСТИКА ПРОГРАММЫ

Программа используется как библиотека для языка Python. Входной точкой является вызов функций планирования для созданных в программе внутренних представлений вычислительной модели и задач.

В ходе работы программы производится запись протокола в консоль и в соответствующий файл (./logs/log.log). Ошибки, связанные с планированием, выводятся вместе с описанием этапа возникновения. Ошибки, связанные с некорректностью входных данных и повлекшие вызов исключения, не обрабатываются.

При неуспешном завершении необходимо исправить ошибки и запустить программу с новыми данными. При отказе системы требуется заново запустить программу после восстановления работоспособности системы.

3. ОБРАЩЕНИЕ К ПРОГРАММЕ

Программа используется как библиотека для языка Python. Входной точкой является вызов функций планирования для созданных в программе внутренних представлений вычислительной модели и задач.

4. ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ

Входными и выходными данными являются объекты в системе Python, реализующие классы внутреннего представления алгоритма планирования. Входные данные передаются через именованные аргументы в функции планирования. Результат планирования возвращается из функции в основную программу.

Способ загрузки входных данных и обработки результата определяется пользователем.

5. СООБЩЕНИЯ

Специальных сообщений об ошибках не предусмотрено. При возникновении исключений времени исполнения системы Python или получении некорректного результата необходимо проконсультироваться с автором продукта, предоставив входные данные и/или протокол работы для запуска с ошибками.

Лист регистрации изменений									
Номера листов (страниц)					Всего листов (страниц) в докум.	№ документа	Входящий № сопроводительного докум. и дата	Подп.	Дата
№ изм.	измененных	замененных	новых	аннулированных					

ПРИЛОЖЕНИЕ Г

Задача вычисления чисел Фибоначчи

Постановка задачи

Задано рекуррентное соотношение для чисел: $F_1 = c_1$, $F_2 = c_2$, $F_n = F_{n-1} + Fn - 2, n > 2$. Также определим сумму первых n чисел Фибоначчи: $S_n = c_3 + \sum_{i=1}^{n} F_i$.

В таком случае вычислительная модель этой предметной области может выглядеть так (рисунок Г.1).

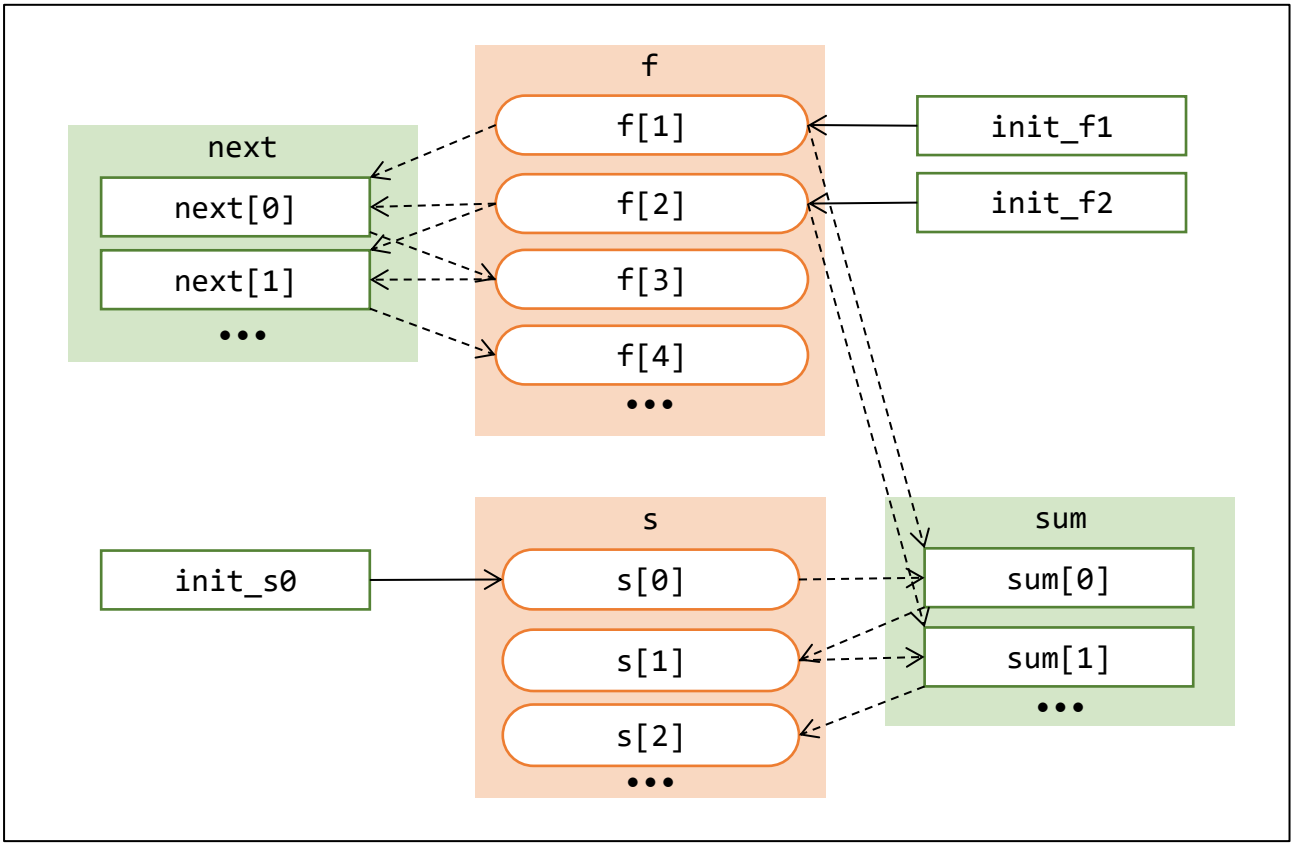

Рисунок Г.1 – Вычислительная модель с числами Фибоначчи

Здесь константы c_1, c_2, c_3 задаются операциями *init_f1*, *init_f2*, *init_s0* соответственно. Сумма вычисляется рекурсивно: $S_n = S_{n-1} + F_n$.

Общая задача на ВМ формулируется так: входные данные отстутствуют; требуется найти все числа Фибоначчи и все частичные суммы. Частная задача при тестировании была поставлена следующая: найти 100-ое число Фибоначчи и частичные суммы для 10,13,16,19 первых чисел. Ожидалось автоматическое

попадание в план всех операций, вычисляющих промежуточные числа до 100-ого, а также только операций частичных сумм только до 18-й включительно.

Программа, выполняющая планирование для такой задачи, с использованием библиотеки может быть записана так, как показано на рисунке Г.2.

При запуске системы планирования были сконструированы две пси-операции. *psi0* задаёт вычисление из первых двух чисел всех остальных, а *psi1* задаёт вычисление всех сумм, учитывая что может быть вычислено любое число Фибоначчи. Результат (внутренне представление промежуточной ВМ) представлен на рисунке Г.3. Зеленым цветом выделены множества термов, описываемых пси-операциями.

Видно, что терм, описываемый *psi1* использует более сложную зависимость, чем ожидалась. Это связано с порядком обработки операций на этапе имитации планирования – до конструирования *psi0* термы с операцией *sum* не были самоподпитывающимися. В данном случае это может привести к появлению лишних итераций при планировании на частной задаче.

При дальнейшем планировании на частной задаче было получено, что для вычисления **W** необходимо выполнить 98 вхождений операций *next*. А вот вместо необходимых 19 вхождений *sum* в план попало 20, как раз из-за более сложного параметрического терма. Результат представлен на рисунке Г.4. Зелёным цветом выделены множества определения операций.

Данный пример расширяет задачу на клеточного-автоматной модели FHP-MP, рассмотренную ранее в работе, тем, что здесь уже можно совершать параллельное вычисление в два потока.

```python
# from {...} import *

# псевдонимы для сокращения записей
R = Range
VS = VariablesSet

modelOperations=[
    SOperation('initf1',
               ins=[] ,
               outs=[('f',[R(1)])]),
    SOperation('initf2',
               ins=[] ,
               outs=[('f',[R(2)])]),
    SOperation('inits',
               ins=[] ,
               outs=[('s',[R(0)])]),
    MOperation('next',
               ins=[('f',(1,1)),('f',(1,2))],
               outs=[('f',(1,3))],
               scope=VS([('next',[R((1,0))])])),
    MOperation('sum',
               ins=[('f',(1,1)),('s',(1,0))],
               outs=[('s',(1,1))],
               scope=VS([('sum',[R((1,0))])]))
    ]

modelVariables=[
    MVariable('f',
              entry=['next', 'sum'],
              comp=['initf1','initf2','next']),
    MVariable('s',
              entry=['sum'],
              comp=['inits','sum'])
]
fib_model = Model(v={x.label : x for x in modelVariables},
              o={x.label : x for x in modelOperations})

# планирование на общей задаче
fib_model_general = model_plan(fib_model,
                               V=VS(),
                               W=VS([('f',[R((1,1))]),('s',[R((1,0))])])
                               )

# планирование на частной задаче
fib_model_specific = model_fit(fib_model, fib_model_general,
                               V=VS(),
                               W=VS([('s',[R(10,20,3)]), ('f',
[R(100)])])
                               )
```

Рисунок Г.2 – Использование библиотеки при планировании для задачи вычисления чисел Фибоначчи

```txt
{
    'f': f(
        entry:['psi0', 'psi1'],
        comp:['initf1', 'initf2', 'psi0']
    ),
    's': s(entry:['psi1'], comp:['inits', 'psi1'])
}
initf1: initf1(ins:[], outs:[('f', [{1}])])

initf2: initf2(ins:[], outs:[('f', [{2}])])

inits: inits(ins:[], outs:[('s', [{0}])])

psi0: psi0(psi: next[(1, 0)](f[(1, 1)], f[(1, 2)]))(
    ins:{
        f: {1,2}
    }, outs:{
        f: Range(1,3)
    })

psi1: psi1(psi:
        sum[(1, 1)](f[(1, 2)], sum[(1, 0)](f[(1, 1)], s[(1, 0)]))
    )(
    ins:{
        f: Range(1,1);
        s: {0}
    }, outs:{
        s: Range(1,1)
    })
```

Рисунок Г.3 – Внутреннее представление плана для общей задачи вычисления чисел Фибоначчи

```txt
{
    'f': f(
        entry:['next', 'sum'],
        comp:['initf1', 'initf2', 'next']
    ),
    's': s(entry:['sum'], comp:['inits', 'sum'])
}
initf1: initf1(ins:[], outs:[('f', [{1}])])

initf2: initf2(ins:[], outs:[('f', [{2}])])

inits: inits(ins:[], outs:[('s', [{0}])])

next: next[{
    next:   {0,1,...,97}(step=1)
}](
    ins:[('f', (1, 1)), ('f', (1, 2))],
    outs:[('f', (1, 3))]
  )

sum: sum[{
    sum:    {0,1,...,19}(step=1)
}](
    ins:[('f', (1, 1)), ('s', (1, 0))],
    outs:[('s', (1, 1))]
  )
```

Рисунок Г.4 – Внутреннее представление плана для частной задачи вычисления чисел Фибоначчи