РАЗРАБОТКА ДЕКЛАРАТИВНОГО СРЕДСТВА ЧИСЛЕННОГО МОДЕЛИРОВАНИЯ ДЛЯ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ. СПЕЦИФИКАЦИЯ ПРЕОБРАЗОВАНИЯ И ЗАДАЧИ. АЛГОРИТМ ПОИСКА ПРИМЕНИМОГО МЕТОДА РЕШЕНИЯ

Р.А. Ескин[1], И.Л. Артемьева[2]

[1] Институт прикладной математики ДВО РАН, Владивосток
[2] Дальневосточный федеральный университет, Владивосток

Введение

Моделирование в современном мире применяется для широкого круга задач, таких как анализ распространения загрязняющих веществ в атмосфере, прогнозирование погоды, в области космических исследований и в других областях науки и техники. Вычислительные модели требуют выполнения больших объемов вычислений. Например, для проведения одного численного эксперимента с глобальной моделью атмосферы необходимо выполнить порядка 10^{16}—10^{17} арифметических операций с плавающей запятой [1]. Учитывая, что проведение таких экспериментов в ходе исследовательской деятельности производится не единожды, становится очевидным актуальность решения проблемы ускорения вычислений.

На данный момент, очевидно, что использование параллельных вычислений является единственным способом повышения мощности вычислительной системы. Дальнейшее серьезное увеличение быстродействия однопроцессорных компьютеров только за счет совершенствования элементной базы становится принципиально невозможным. Этому препятствуют три основных физических барьера: световой, тепловой и квантовый [4].

Но на практике почти всегда справедливо утверждение: "повышение степени параллелизма в архитектуре компьютера ведет как к росту его пиковой производительности, так одновременно и к увеличению разрыва между производительностью пиковой и реальной". Для того чтобы получить реальное улучшение производительности на параллельной вычислительной системе по сравнению с последовательной, необходимы значительные модификации программы, требующие от специалистов, занимающихся исследованиями в области моделирования, знаний и умений специфичного характера (таких как знание языка программирования, детального понимания архитектуры вычислительной системы). Адаптация программы к параллельным вычислениям является трудозатратным и сложным процессом, который отвлекает ресурсы от непосредственных исследований. Решением могло бы стать создание программной системы, автоматизирующей процесс распараллеливания программ.

Использование императивных языков усложняет процесс распараллеливания в силу необходимости тщательного анализа зависимостей. Декларативный подход, в свою очередь, успел себя хорошо зарекомендовать в таких языках как НОРМА[2], РЕПРО-С[3]. Использование современных декларативных языков позволяет снизить трудозатраты на создание модели при сохраненной производительности, но обязывает пользователя выполнить преобразование математической модели к некоторому виду (например, к разностной схеме) или изначально ограничивает круг возможных решаемых задач. Очевидно, создание программной системы, позволяющей задать описание в терминах математической модели и поддерживающей расширяемое множество методов решения, ориентированных на параллельные вычислительные системы, является актуальным.

Целью данной работы является описать структуру декларативного средства численного моделирования для параллельных вычислительных систем, модель декларативного представления методов численного решения задач математического моделирования, модель спецификации задачи для данной системы, а также алгоритм поиска применимого метода решения.

Структура программного средства

Разрабатываемая программная система имеет следующую структуру (Рис. 1): на вход системе подается описание модели на декларативном языке спецификации, производится лексический и синтаксический анализ модели и строится ее внутреннее представление. По внутреннему представлению задачи происходит поиск подходящего метода решения задачи. В системе планируется поддержка расширяемого множества методов численного решения. Это будет обеспечено за счет наличия банка преобразований. Преобразование представляет собой описание решения задачи и описание условий применимости данного решения. Эффективность каждого конкретного метода решения зависит от задачи и специфики доступной вычислительной системы. Задача преобразования – поставить в соответствие задаче наиболее эффективный метод её решения. Более подробно структура преобразования будет описана ниже.

Инновацией в архитектуре системы является блок анализа задачи. Его назначение – поиск и интерпретация применимого преобразования, то есть выбор применимого преобразования из банка преобразований и составление решателя задачи из преобразования и спецификации задачи. Выбор применимого

преобразования из банка преобразований является задачей перебора – контекстное условие каждого преобразования из банка должно быть интерпретировано и сравнено со спецификацией задачи. Если сравнение после интерпретации успешно, тогда преобразование применимо. Интерпретация и сравнение производятся с использованием деревьев разбора спецификации задачи и контекстного условия.

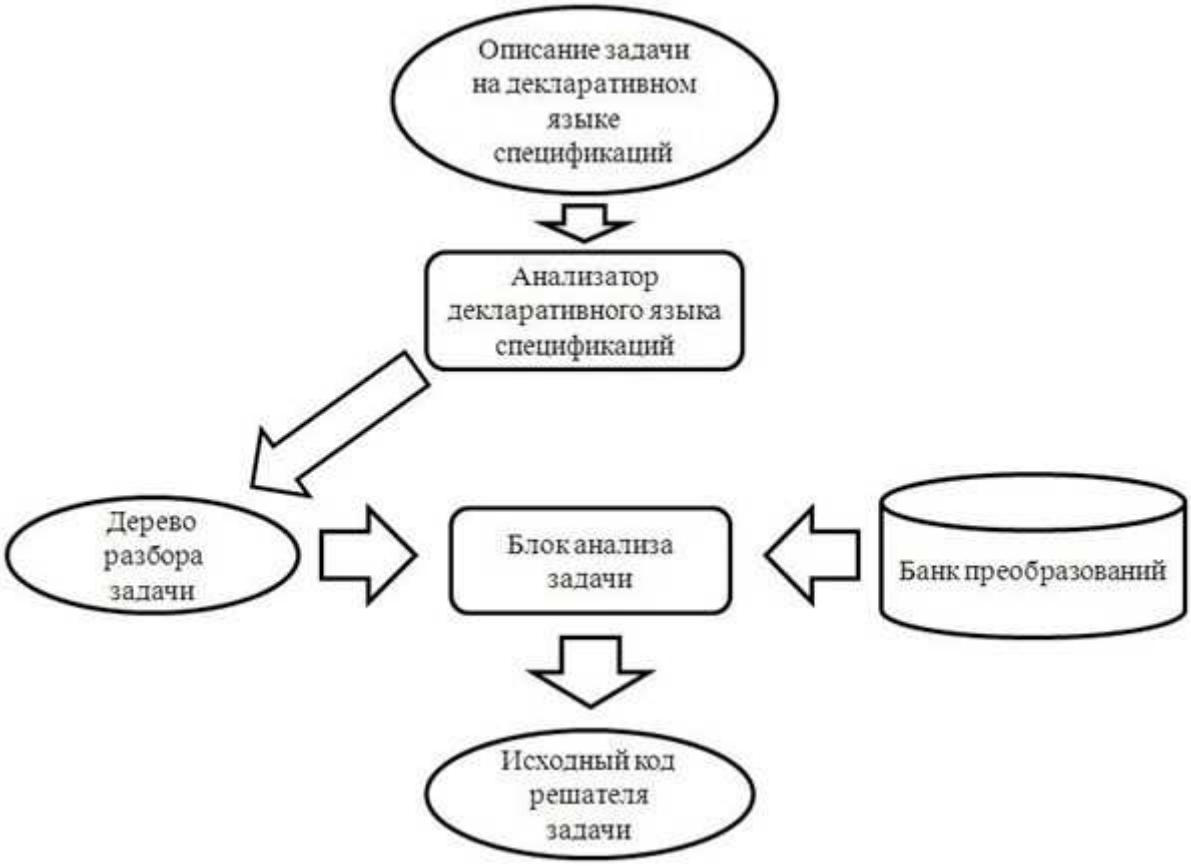

Рис. 1. Архитектура программной системы

Описание задачи

Спецификация задачи содержит:

1. модель задачи,
2. спецификацию входных и выходных данных,
3. спецификацию дополнительных свойств задачи.

Модель задачи содержит описание основных и вспомогательных имен, а также предложения, описывающие ограничения на интерпретацию имен модели и представляется прикладной логической теорией на языке прикладной логики[5].

Спецификация входных и выходных данных определяет, какие из имен прикладной логической модели являются исходными данными задачи, а какие – результатом решения задачи. Спецификация входных и выходных данных задается в виде пар, состоящих из имени и значения или источника, откуда это значения может быть взято (в случае входных данных) или куда значение должно быть помещено после решения задачи (для выходных данных). В качестве таких источников могут выступать имена файлов, имена баз данных и т. п. Для любой спецификации задачи истинно то, что объединение множеств входных и выходных имен данной спецификации является подмножеством всех имен данной спецификации.

Спецификация дополнительных свойств задачи представляет собой множество пар имя-значение. Дополнительные свойства задачи задают те свойства, которые напрямую не следуют из структуры задачи, но могут влиять на выбор метода решения, как, например, целевая платформа для решателя задачи.

Рассмотрим пример задачи и ее спецификации для задачи о клике. Кликой в неориентированном графе называется подмножество вершин, каждые две из которых соединены ребром графа. Иными словами, это полный подграф первоначального графа. Требуется найти в заданном графе клику максимального размера

Опишем модель задачи. Для этого необходимо ввести специализированное расширение языка прикладной логики Графы:

1. Терм GRAPH означает множество всех возможных графов.
2. Терм VERTEXES_SET означает множество всех возможных вершин графов.
3. Терм EDGES_SET означает множество всех возможных дуг графов.
4. Терм Подграфы(G), где G –элемент множества GRAPH, обозначает множество всех подграфов графа G.

5. Терм VERTEXES (G), где G – элемент множества GRAPH, обозначает множество вершин графа G.
6. Терм EDGES(G), где G – элемент множества GRAPH, обозначает множество дуг графа G.
7. Терм EDGE (X,Y), где X,Y –элементы множества VERTEXES_SET, обозначает дугу между X, Y.
Спецификация задачи о клике:
1. Прикладная логическая теория Задача о клике (ST, Графы):
Основные термины модели:
сорт Исходный граф: GRAPH;
сорт Клики: {} SUBGRAPHS(Исходный граф);
сорт Максимальная клика: SUBGRAPHS(Исходный граф);
Ограничения на интерпретацию имен модели:
(var_x:SUBGRAPHS(Исходный граф))
(var_x1: VERTEXES(var_x))
(var_x2: VERTEXES(var_x))
(var_x∈Клики)↔(EDGE(var_x1, var_x2) ∈ EDGES(Исходный граф));
(var_x:Клики)
(Максимальная клика ∈ Клики) & (μ(VERTEXES(Максимальная клика)) >= μ (VERTEXES(var_x))).
2. Спецификация входных и выходных данных:
a. входные данные:
i. Исходный граф: input.txt
b. выходные данные
i. Клики: output.txt
ii. Максимальная клика: output.txt
3. Спецификация дополнительных свойств задачи: пусто.

Описание банка преобразований

Для того чтобы поставить в соответствие задаче наиболее эффективный метод её решения, используется преобразование. Преобразование состоит из контекстного условия (КУ) и набора трансформаций. К одной задаче может быть применим более чем один метод решения, каждому методу соответствует своя трансформация. Контекстное условие представляет собой схему задачи и определяет применимость преобразования к некоторой задаче. КУ состоит из шаблона задачи, спецификации входных и выходных имен, и дополнительных условий. Шаблон задачи определяет общую структуру задачи, к которой может быть применено данное преобразование. Спецификация входных и выходных имен определяет, какие из имен, использованных в шаблоне задачи, являются входными и выходными данными задачи без конкретизации их значений. Шаблон задачи задается с помощью модифицированного языка прикладной логики, причем все имена, входящие в КУ, считаются абстрактными. В ходе поиска применимого преобразования будет произведена конкретизация этих имен – каждому из них будет поставлено в соответствие конкретное имя из задачи. Подробнее алгоритм поиска применимого преобразования описан в соответствующем подразделе. Применимость метода решения не всегда следует только из структуры задачи – может потребоваться дополнительный анализ свойств задачи. Поэтому круг решаемых задач ограничивают также дополнительные условия. Дополнительные условия задаются в виде множества пар имя-значение. В каждом преобразовании может содержаться более одной трансформации. Выбор трансформации из набора зависит от свойств решаемой задачи (например, размерности).

Трансформация представляет собой пару <F, S>, где F – функция эффективности трансформации, S – схема решения.

Функция эффективности представляет собой вещественную положительную функцию, зависящую от свойств решаемой задачи. Функция эффективности используется для выбора трансформации из набора – чем больше значение функции для конкретной задачи, тем эффективней связанная с ней схема.

Схема решения представляет собой решатель задачи, записанный с помощью конкретного языка программирования для параллельной вычислительной системы. Часть имен переменных и функций в записи схемы решения могут являться абстрактными символами, которые будут конкретизированы после интерпретации контекстного условия. Множество абстрактных имен трансформации должно быть подмножеством абстрактных имен контекстного условия для обеспечения применимости интерпретации контекстного условия к трансформации.

Алгоритм поиска применимого преобразования

Алгоритм поиска применимого преобразования по заданной спецификации задачи из множества преобразований использует в своей работе деревья синтаксического разбора задачи и контекстного условия преобразования. Спецификация задачи и контекстное условие задаются в декларативном виде с помощью языка прикладной логики [5]. Для лексического и синтаксического разбора описания задачи и контекстного условия, записанных на ЯПЛ, может использоваться программное средство Pally [6, 7]. Деревья синтаксического разбора подаются на вход алгоритму. На выходе алгоритма – решатель задачи на некотором языке программирования, полученный в результате применения к трансформации применимого преобразования подстановки имен,

найденной в ходе работы алгоритма. Алгоритм является переборным и состоит из следующих шагов (применяемых для каждого преобразования из банка преобразований):

1. Поиск глобальной подстановки имен.

2. Поиск поддерева в дереве задачи, эквивалентного дереву контекстного условия после применения подстановки, найденной на шаге 1.

3. Если такое поддерево существует, то данное преобразование применимо, и найденная глобальная подстановка имен можно применяется к трансформации преобразования.

Последний пункт осуществляется тривиальной заменой имен в трансформации на соответствующие имена из подстановки.

Для формального определения алгоритма сопоставления деревьев синтаксического разбора задачи и контекстного условия, введем следующие обозначения:

TASK_TREE– дерево синтаксического разбора прикладной логической теории.

CC_TREE – дерево синтаксического разбора контекстного условия преобразования.

ROOT(T) – корень дерева T.

GLOBAL(T) – множество поддеревьев T, задающих сорта/значения имен. Корень каждого из таких поддеревьев имеет метку, принадлежащую множеству {".SORT_NAME", ".VALUE_NAME"}.

NAME(subT) – функция, определенная на GLOBAL(T), возвращающая имя subT.

SORT(subT) – функция, определенная на GLOBAL(T), возвращающая сорт subT.

FORMULA (T) – множество поддеревьев T, задающих предложении-связи между именами. Для каждого элемента множества FORMULA(T) существует единственное поддерево данного элемента, корень которого имеет метку со значением ".FORMULA".

PREFIX (T_1) – множество поддеревьев дерева T_1 (где T_1 принадлежит FORMULA (T)), задающих локальные переменные формулы. Корень каждого из таких поддеревьев имеет метку со значением ".VAR_DESCRIPT".

GLOBAL_NAMES(T) – множество глобальных имен (T – дерево синтаксического разбора некоторой прикладной логической теории).

LOCAL_NAMES(T,T_1) – множество локальных имен предложения T_1 (T – дерево синтаксического разбора некоторой прикладной логической теории, T_1 принадлежит FORMULA (T)).

Глобальная подстановка имен из дерева A в дерево B- конечное множество пар имен <$name_1$, $name_2$>, где $name_1$ принадлежит GLOBAL_NAMES(A), $name_2$ принадлежит GLOBAL_NAMES(B) (A, B – деревья синтаксического разбора некоторых прикладных логических теорий).

Локальная подстановка имен из поддерева A_1 дерева A в поддерево B_1 дерева B - конечное множество пар имен <$name_1$, $name_2$>, $name_1$ принадлежит LOCAL_NAMES (A, A_1), $name_2$ принадлежит GLOBAL_NAMES(B, B_1), (A, B – деревья синтаксического разбора некоторых прикладных логических теорий).

Значение глобальной подстановки(name)– такое имя $name_2$, что пара<name, $name_2$>принадлежит глобальной подстановке имен.

Значение локальной подстановки(name)–такое имя $name_2$, что пара<name, $name_2$>принадлежит локальной подстановке имен.

Алгоритм поиска глобальной подстановки имен:

Вход алгоритма: CC_TREE, TASK_TREE.

Описание алгоритма:

Шаг 1. Взять следующее поддерево X дерева CC_TREE, если новых поддеревьев больше не осталось – перейти на шаг 6.

Шаг 2. Если X принадлежит GLOBAL (CC_TREE) то перейти на шаг 3, иначе перейти на шаг 1.

Шаг 3. Взять следующее непомеченное поддерево Y дерева TASK_TREE, если новых поддеревьев больше не осталось – перейти на шаг 5.

Шаг 4. Если Y принадлежит GLOBAL (TASK_TREE) и SORT(X)==SORT(Y), то пометить Y и добавить пару <NAME (X), NAME (Y)> в глобальную подстановку и перейти к шагу 1, иначе перейти на шаг 3.

Шаг 5. Неуспешное завершение алгоритма.

Шаг 6. Успешное завершение алгоритма.

Алгоритм поиск поддерева в дереве задачи, эквивалентное дереву контекстного условия после применения глобальной подстановки:

Входа алгоритма: CC_TREE, TASK_TREE.

Описание алгоритма:

Шаг 1. Взять следующее поддерево X дерева CC_TREE, если новых поддеревьев больше не осталось – перейти на шаг 7.

Шаг 2. Если X принадлежит FORMULA (CC_TREE), то перейти на шаг 3, иначе на шаг 1.

Шаг 3.Взять следующее непомеченное поддерево Yдерева TASK_TREE, если новых поддеревьев больше не осталось – перейти на шаг 6.

Шаг 4. Если Y принадлежит FORMULA (TASK_TREE) , то перейти на шаг 5, иначе на шаг 3.

Шаг 5. Найти локальную подстановку (алгоритм приведен ниже) и если TRUE==COMPARE(X,Y), то пометить Yи перейти на шаг 1, иначе на шаг 3.

Шаг 6. Неуспешное завершение алгоритма.

Шаг 7. Успешное завершение алгоритма.

Алгоритм поиска локальной подстановки:

Входа алгоритма: X — элемент множества FORMULA(CC_TREE), Y — элемент множества FORMULA (TASK_TREE)

Описание алгоритма:

Шаг 1. Взять следующее поддерево X_1дерева X, если новых поддеревьев больше не осталось – перейти на шаг 6.

Шаг 2. ЕслиX_1 принадлежит PREFIX (X), то перейти на шаг 3, иначе на шаг 1.

Шаг 3. Взять следующее непомеченное поддерево Y_1 дерева Y,если новых поддеревьев больше не осталось – перейти на шаг 5.

Шаг 4. ЕслиY_1 принадлежит PREFIX(Y) и SORT(X)==SORT(Y), то пометить Y_1и добавить пару <NAME (X_1), NAME (Y_1)>в локальную подстановку и перейти к шагу 1, иначе перейти к шагу 3.

Шаг 5. Неуспешное завершение алгоритма.

Шаг 6. Успешное завершение алгоритма.

В описании алгоритмов использовались функции COMPARE и EQUAL. Их определение дано ниже.

Определение рекурсивной функции COMPARE:

Вход функции: A - дерево синтаксического разбора прикладной логической теории, B - дерево синтаксического разбора прикладной логической теории.

Описание функции:

Шаг 1. Если A – лист, и B – лист, то перейти на шаг 2, иначе на шаг 3.

Шаг 2. Если EQUAL(A,B), то перейти на шаг 8, иначе перейти на шаг 7.

Шаг 3. Если EQUAL(ROOT (A) , ROOT (B)), то перейти на шаг 4, иначе на шаг 7.

Шаг 4. Взять следующее поддерево A_1 дерева A, если новых поддеревьев больше не осталось – перейти на шаг 8.

Шаг 5. Взять следующее непомеченное поддерево B_1 дерева B, если новых поддеревьев больше не осталось – перейти на шаг 7.

Шаг 6. Если COMPARE(A_1, B_1), то пометить B_1и перейти на шаг 4.

Шаг 7. Неуспешное завершение процедуры. Возвратить FALSE.

Шаг 8. Успешное завершение процедуры. Возвратить TRUE.

Определение функции EQUAL:

Вход функции: A – лист дерева синтаксического разбора прикладной логической теории, B–лист дерева синтаксического разбора прикладной логической теории.

Описание функции:

Шаг 1. Если A==B или A== Значение глобальной подстановки(B) или A== Значение локальной подстановки(B), то перейти на шаг 3, иначе на шаг 2.

Шаг 2. Неуспешное завершение процедуры. Возвратить FALSE.

Шаг 3. Успешное завершение процедуры. Возвратить TRUE.

Пример работы алгоритма сопоставления дерева задачи и дерева контекстного условия.

Рассмотрим задачу о клике, модель которой представлена выше. Дерево разбора прикладной логической теории для этой задачи имеет вид:

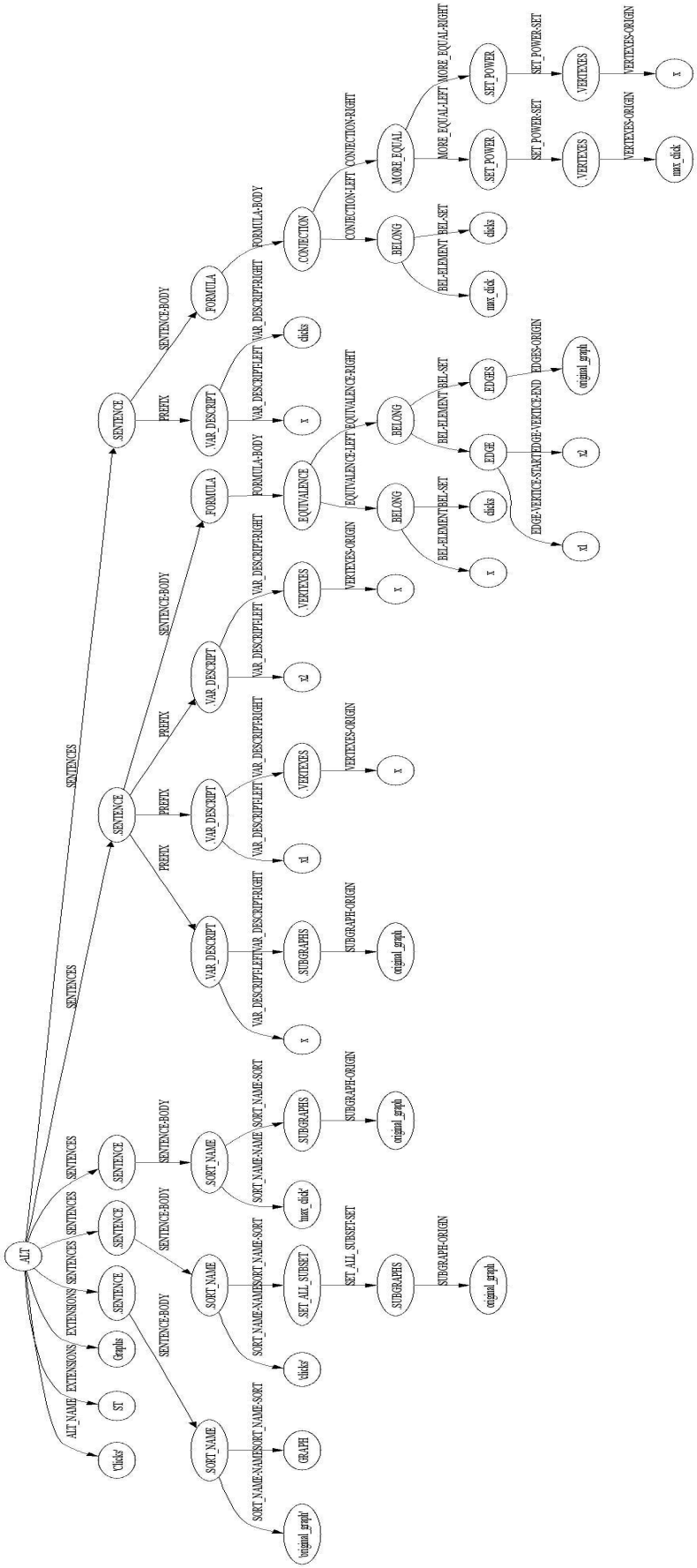

Рис. 2. Дерево разбора ПЛТ задачи

Прикладная логическая теория контекстного условия применимой к этой задаче трансформации, записанная на ЯПЛ:

NAME 'context_condition_Clicks';

EXTENSIONS ST, Graphs;
SENTENCES
HY('object_graph') = GRAPH;
HY('set_clicks')= {} SUBGRAPHS(object_graph);
HY('object_max_click')=SUBGRAPHS(object_graph);
(var_x: set_clicks)
(object_max_click BEL set_clicks) AND (MU(VERTEXES(object_max_click)) >= MU(VERTEXES(var_x))).

Дерево синтаксического разбора прикладной логической теории контекстного условия имеет вид:

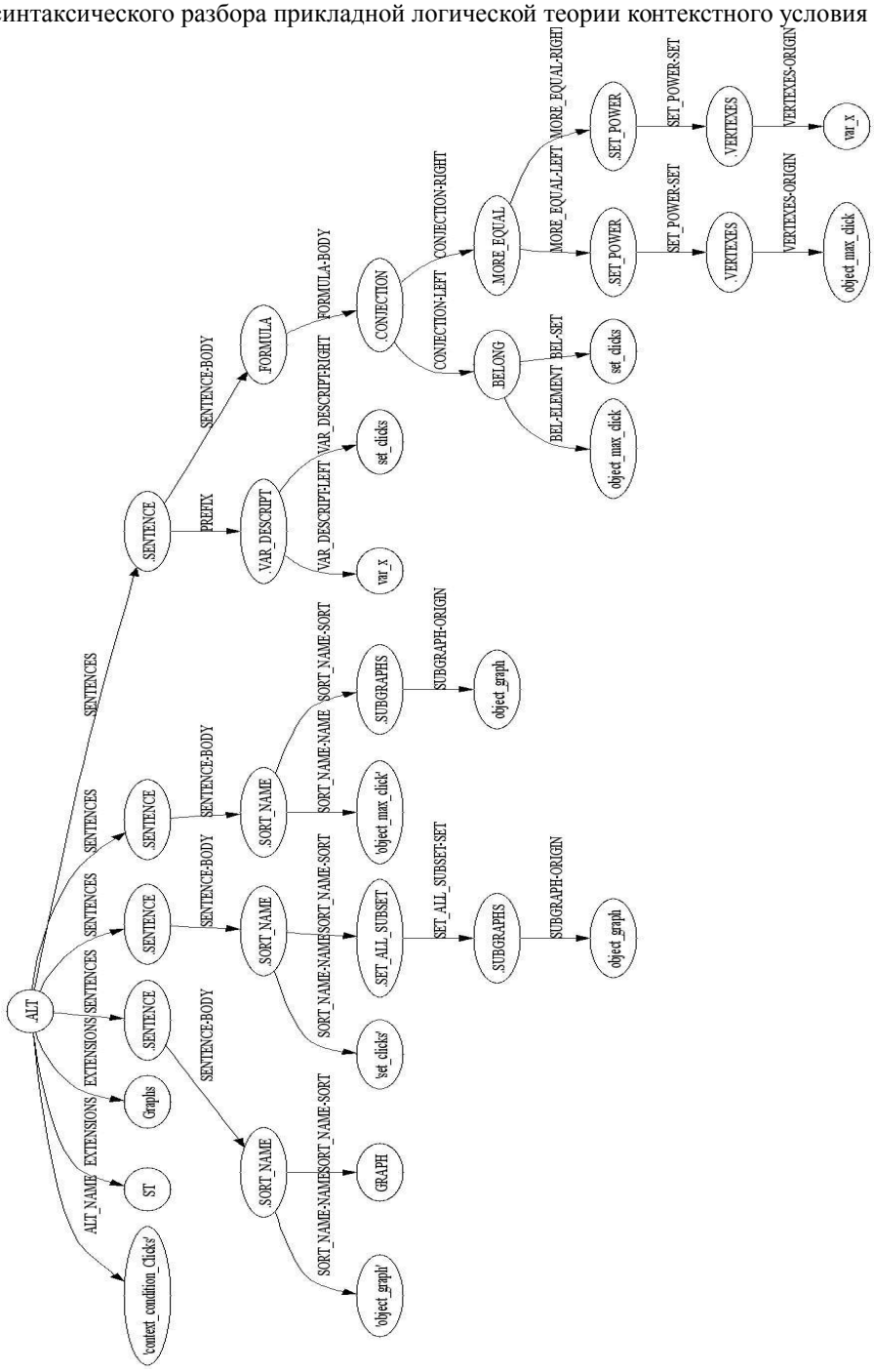

Рис. 3. Дерево разбора ПЛТ контекстного условия

Оба дерева подаются на вход алгоритму поиска применимого преобразования. Алгоритм ищет поддерево в дереве задачи, эквивалентное дереву контекстного условия. Найденное поддерево в дереве задачи, эквивалентное дереву контекстного условия после применения подстановки, выделено в структуре общего дерева задачи на рисунке 4.

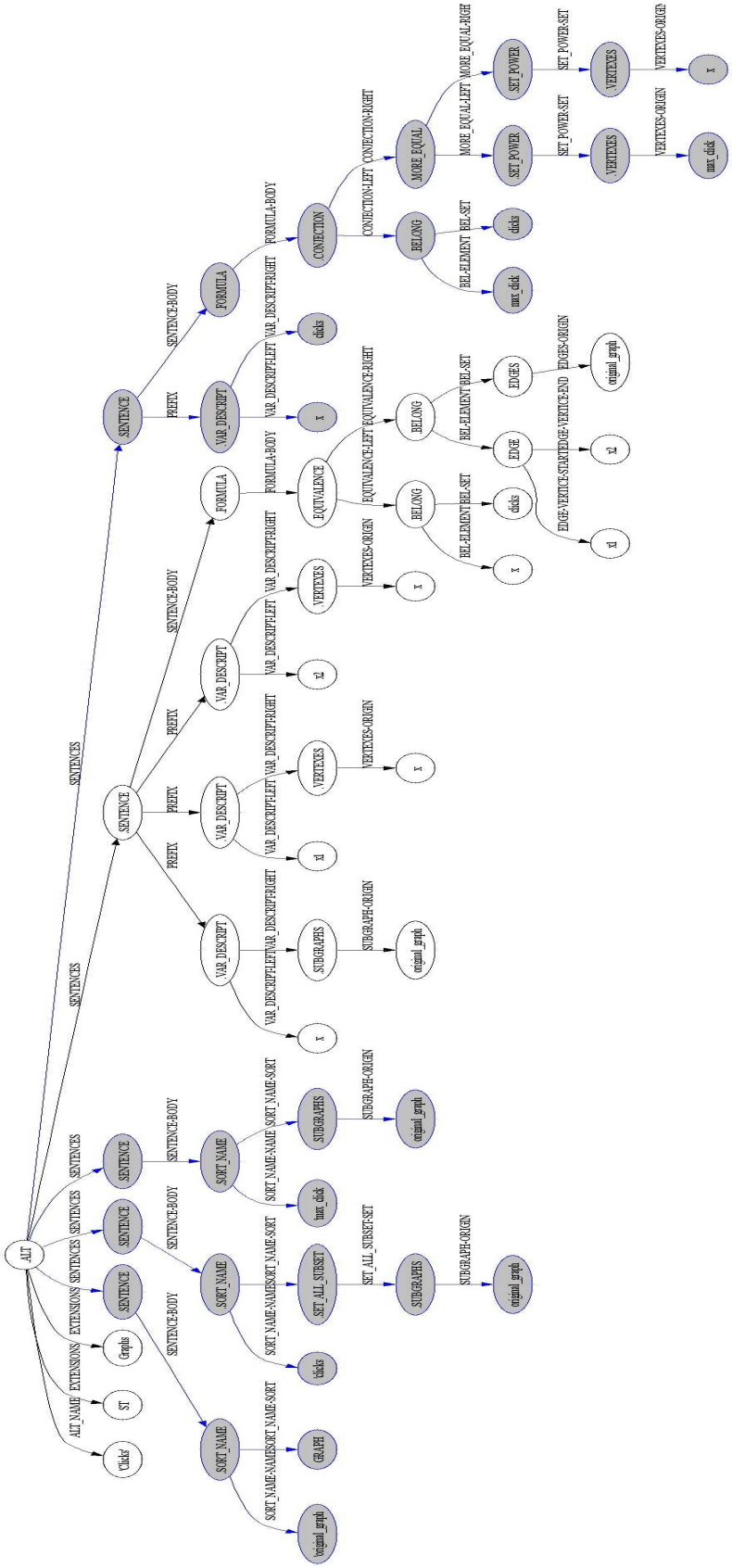

Рис. 4. Визуализация найденного поддерева контекстного условия в дереве задачи

Как было сказано выше, последний шаг алгоритма (применение глобальной подстановки имен к трансформации преобразования) осуществляется тривиальной заменой имен в трансформации на соответствующие имена из подстановки, и поэтому данный шаг не будет расписан.

Заключение

В работе рассмотрена модель описания задачи на декларативном языке спецификации и модель декларативного представления методов решения задач математического моделирования. Описана концепция программной системы, позволяющей задать описание в терминах математической модели и поддерживающей расширяемое множество методов решения, ориентированных на параллельные вычислительные системы. Представлен алгоритм поиска применимого преобразования по заданной спецификации задачи из множества преобразований и приведен пример работы алгоритма. Данный алгоритм является основой разрабатываемой программной системы.

ЛИТЕРАТУРА:

1. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – СПб.: БХВ-Петербург, 2002.
2. Андрианов А.Н., Ефимкин К.Н., Задыхайло И.Б. Язык Норма. Препринт ИПМ им. М.В.Келдыша АН СССР, #165, 1985.
3. Тютюнник М.Б. "Разработка и исследование продукционной системы параллельного программирования": автореф… дис. канд. техн. наук. – Владивосток: Институт автоматики и процессов управления ДВО РАН, 2010.
4. Юфрякова О.А. Препятствия к неограниченному масштабированию параллельных систем // Материалы Международной суперкомпьютерной конференции "Научный сервис в сети Интернет: экзафлопсное будущее". – Новороссийск, 2011. – С. 267-270.
5. Клещёв А.С., Артемьева И.Л. Необогащенные системы логических соотношений // Научно-техническая информация . – 2000 . – № 7 . – С.18-28.
6. Процессор класса языков прикладной логики. Способ описания языка [Электронный ресурс]. – Режим доступа: http://www.iacp.dvo.ru/is/publications/Kleschev,Smagin-Pally_Language.pdf
7. Процессор класса языков прикладной логики. Метод реализации [Электронный ресурс]. – Режим доступа: http://www.iacp.dvo.ru/is/publications/Kleschev,Smagin-Pally_Realization.pdf