На правах рукописи

Тютюнник Михаил Борисович

РАЗРАБОТКА И ИССЛЕДОВАНИЕ ПРОДУКЦИОННОЙ СИСТЕМЫ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ

05.13.11 – математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

Автореферат
диссертации на соискание ученой степени
кандидата технических наук

Владивосток
2010

Работа выполнена в лаборатории интеллектуальных систем Института автоматики и процессов управления Дальневосточного отделения РАН.

Научный руководитель	доктор технических наук Артемьева Ирина Леонидовна
Официальные оппоненты	доктор физико-математических наук, профессор Касьянов Виктор Николаевич
	кандидат технических наук Харитонов Дмитрий Иванович
Ведущая организация	Московский Энергетический Институт (Технический Университет), г. Москва

Защита состоится “11” февраля 2011 г. в 12 часов на заседании диссертационного совета Д 005.007.01 при Институте автоматики и процессов управления ДВО РАН по адресу: 690041, г. Владивосток, ул. Радио, д.5.

С диссертацией можно ознакомиться в библиотеке Института автоматики и процессов управления Дальневосточного отделения РАН.

Автореферат разослан «28» декабря 2010 г.

Ученый секретарь
диссертационного совета Д 005.007.01
к.т.н. _____ А.В. Лебедев

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Актуальность проблемы. Продукционные языки и системы, основанные на правилах, являются важной технологией при создании информационных систем, называемых системами с базами правил (или с базами знаний, представленными в виде правил). Представление задачи в виде множества правил является более естественным способом по сравнению с алгоритмом и не требует от разработчика программной системы специальных знаний по организации вычислительного процесса: управление этим процессом реализует языковой процессор языка, основанного на правилах. При использовании систем продукций для решения реальных прикладных задач к настоящему времени разработаны базы знаний объемом в тысячи правил. В современных проектах по созданию Семантического Интернета также предполагается использование правил как средств, расширяющих возможности языка для представления онтологий OWL (Web Ontology Language) и позволяющих описывать бизнес-логику программных систем, создавать адаптеры между информационными системами и определять различные сложные запросы к информационным компонентам программных систем.

Если в системе, основанной на правилах, результат решения задачи зависит от порядка применения правил, в механизме вывода неявно присутствует управление выбором правил при решении задачи либо в систему продукций вводятся явные средства для задания управления (т.е. для задания алгоритма управления). В системах конфлюэнтных[1] продукций результат логического вывода не зависит от порядка применения правил, что не требует введения явных или неявных средств управления процессом решения задачи, позволяя рассматривать конфлюэнтные системы продукций не только как средство для представления знаний, но и как средство для задания метода решения задачи, т.е. как язык программирования метода. Поэтому системы конфлюэнтных продукций являются важным классом продукций. На основе модели конфлюэнтных продукций были созданы системы СИНАП и РЕПРО, которые использовались для построения многих прикладных экспертных систем.

Существует два основных способа организации вывода в системах продукций: прямой и обратный. При прямом выводе система использует информацию из антецедента правил, чтобы вывести информацию, описанную в консеквенте правил. При обратном выводе выдвигается цель - установить, можно ли вывести некоторый факт из системы продукций. Чем больше правил содержит система продукций, тем медленнее выполняется логический вывод, поэтому исследовались различные методы ускорения вывода. К настоящему времени разработаны методы оптимизации как для прямого, так и для обратного методов организации вывода, в том числе для конфлюэнтных систем продукций, причем наибольший эффект при оптимизации получен для систем последнего класса.

Развитие компьютерной архитектуры и сетевых технологий, появление новых научных и прикладных задач, требующих большого объема вычислений, отставание быстродействия существующих последовательных компьютерных комплексов и теоретическая ограниченность роста их производительности привели к необходимости применения многопроцессорных и многоядерных компьютерных систем (МВС), выдвинув параллельные вычисления на одно из центральных мест в современном про-

[1] Конфлюэнтные системы также называются коммутативными или системами, имеющими неподвижную точку.

граммировании и вычислительных технологиях. Это, в свою очередь, потребовало развития языков программирования и создания параллельных языковых процессоров для них. Для систем продукций, реализующих обратный вывод (в частности, систем, основанных на языке Пролог и его модификациях), к настоящему времени описаны методы организации параллельного вывода.

Большой вклад в разработку моделей систем продукций, методов организации процесса логического вывода для таких систем, а также методов распараллеливания вывода для них внесли Д.А. Поспелов, В.Н. Вагин, Г.С. Осипов, И.П. Кузнецов, А.П. Еремеев, А.С. Клещев, В.Л. Стефанюк, А.В. Жожикашвили, Ю.А. Загорулько, Т.М.Яхно, А.С. Нариньяни, S. Vere, D.B. Lenat и многие другие.

В существующих к настоящему времени моделях систем продукций допускаются предметные, функциональные и предикатные имена, причем типы объектов, с которыми могут работать правила, в современных формализмах для представления правил согласованы с типами объектов, используемых в языках для определения онтологий. Однако в существующих программных системах, основанных на правилах, типы данных, используемые для моделирования объектов предметных областей, ограничены, в основном, предикатами, а сами языки являются языками с бедным семантическим базисом. В таких языках отсутствуют средства для компактного представления правил с похожей структурой.

Существующие в настоящее время параллельные программные системы, основанные на правилах, базируются на неконфлюэнтных продукциях, поэтому для них разрабатываются сложные механизмы контроля процесса распараллеленного вывода. При этом при обратном выводе применение алгоритмов распараллеливания затрудняется наличием неявного управления порядком записи правил, порядком записи компонентов правил и порядком записи компонентов целевого утверждения, которое является входом системы, основанной на правилах. Кроме того, наличие неявного управления выбором правил затрудняет сопровождение систем с большим количеством правил. Распараллеливание прямого вывода учитывает лишь очень простые связи между правилами. Методы распараллеливания вывода, ориентированные на конфлюэнтные продукции, которые обладают естественным параллелизмом и не накладывают никаких дополнительных условий на управление процессом вывода, из литературы не известны.

Поэтому актуальными проблемами являются развитие языка системы продукций, а также разработка подходов к распараллеливанию процесса логического вывода для конфлюэнтных систем продукций.

Целью диссертационной работы является разработка и исследование моделей и методов создания системы параллельного программирования на основе модульных конфлюэнтных продукций, в которых язык позволяет использование предметных, функциональных и предикатных имен и ограниченных логических и математических кванторов, имеет средства деления задачи на подзадачи и средства для более экономного представления повторяющихся фрагментов, поддерживаемые наличием модульности и подпрограмм в языках высокого уровня.

Для достижения поставленной цели в диссертационной работе необходимо решить следующие **задачи**:

1. Разработать расширенную модель продукционного языка и на ее основе расширенный язык, основанный на правилах, имеющий средства деления множества правил на модули и более экономного представления повторяющихся фрагментов.

2. Разработать и исследовать схемы распараллеливания процесса логического вывода для системы конфлюэнтных продукций, основанной на разработанной модели языка.
3. Разработать алгоритм управления выбором схем распараллеливания в процессе логического вывода.
4. Разработать методы реализации системы параллельного программирования.
5. Провести экспериментальное исследование системы параллельного программирования на реальных примерах.

Методы исследования. Для решения указанных задач использовались элементы математической логики, теории алгоритмов и исчислений, а также методы системного программирования, объектно-ориентированного программирования, параллельного программирования.

Научная новизна работы состоит в следующем:

- впервые разработана расширенная модель и на ее основе расширенный продукционный язык, особенностью которых является наличие средств для задания схем правил и их конкретизаций, модулей и их вызовов, а также ограниченных кванторов;
- в рамках предложенной модели впервые разработаны схемы распараллеливания процесса логического вывода для системы конфлюэнтных продукций; получены оценки времени исполнения при различных схемах распараллеливания;
- впервые разработан алгоритм управления выбором схем распараллеливания в процессе логического вывода; приведено описание условий и ограничений, налагаемых средой выполнения системы и структурой информационного графа логической программы, учитываемых при выборе схем распараллеливания в процессе выполнения правил.

Практическая ценность работы состоит в том, что разработана система параллельного программирования, которая может быть использована при создании параллельных систем, основанных на правилах, предназначенных для решения задач разных классов на многоядерных персональных компьютерах, а также на кластерах.

Научные результаты диссертационной работы использованы в Дальневосточном государственном университете в учебном процессе при чтении курса лекций по спецдисциплинам «Системы искусственного интеллекта» и «Рекурсивно-логическое программирование» студентам специальности 010503.65 – «Математическое обеспечение и администрирование информационных систем» и выполнении практических заданий по ним.

Результаты работы нашли применение в научных исследованиях сотрудников кафедры ПО ЭВМ Института математики и компьютерных наук ДВГУ и сотрудников лаборатории интеллектуальных систем ИАПУ ДВО РАН.

Получено свидетельство о государственной регистрации программы для ЭВМ №2010614810 «Распараллеливающий компилятор для языка, основанного на правилах». Авторы: Артемьева И.Л., Тютюнник М.Б. Зарегистрировано в Реестре программ для ЭВМ 23 июля 2010 г.

Апробация работы. Основные научные и практические результаты работы докладывались и обсуждались на следующих международных и отечественных конференциях и семинарах: Дальневосточной математической школе-семинаре имени академика Е.В. Золотова (Владивосток, Хабаровск, Находка, 2003-2008), открытом Дальневосточном конкурсе программных средств студентов, аспирантов и молодых специалистов «Программист» (Владивосток, 2004, 2010), II - V международных конференциях «Параллельные вычисления и задачи управления» (Москва, 2004, 2006,

2008, 2010), Международной конференции «Knowledge-Dialog-Solution» (Varna, Bulgaria, 2008), Международной конференции «First Russia and Pacific Conference on Computer Technology and Applications» (Vladivostok, Russia, 2010), Научной сессии МИФИ (Москва, 2006, 2007, 2008), Демидовской конференции «Фундаментальные и прикладные проблемы современной физики» (Москва, 2006), Международной научно-технической конференции «Искусственный интеллект. Интеллектуальные и многопроцессорные системы» (Кацивели, Украина, 2006, 2007), совместных семинарах отдела интеллектуальных систем ИАПУ ДВО РАН и базовой кафедры программного обеспечения ЭВМ ДВГУ при ИАПУ ДВО РАН (2005-2010).

Публикация результатов работы. По материалам диссертации опубликовано 28 печатных работ, из них 2 статьи в журналах, входящих в перечень ВАК (работы [12,15]). Основные публикации приведены в автореферате.

Структура и объём работы. Диссертационная работа состоит из введения, пяти глав, заключения, списка литературы, включающего 100 наименований, и приложений. Основная часть работы изложена на 127 страницах, включая 18 рисунков, 14 диаграмм и 10 таблиц.

СОДЕРЖАНИЕ РАБОТЫ

Глава 1 диссертации содержит обзор литературы. В ней приведен анализ математических основ систем, основанных на правилах, а также дан обзор классов языков, основанных на правилах, рассмотрены существующие методы реализации систем, основанных на правилах, оптимизации и распараллеливания процесса логического вывода для них.

Глава 2 диссертации содержит описание модели расширенного языка, основанного на правилах. При разработке модели расширенного языка были проанализированы созданные в Институте автоматики и процессов управления ДВО РАН модель реляционных конфлюэнтных продукций, модель недоопределенных продукций, модель языка МАКРОРЕПРО, модель языка с ограниченными кванторами и основанные на этих моделях продукционные языки, а также класс языков прикладной логики, предназначенных для определения свойств различных предметных областей и используемых при создании систем, основанных на знаниях.

Модель описывает синтаксис и декларативную семантику языка и определяет термы, формулы, правила, схемы правил, конкретизации схем и вызовы модулей, а также ограниченные кванторы. Операционную семантику языка задают правила логического вывода.

Все используемые в правилах предметные, функциональные и предикатные имена являются терминами предметной области. Все имена имеют сорт, который задается при описании имени. Значение имени может быть точным либо неточным, что задается характеристикой «определенность значения».

Правило (продукция) состоит из префикса и тела. Префикс есть последовательность описаний свободных переменных $(v_1:t_1)(v_2:t_2)...(v_m:t_m)$, где $(v_i:t_i)$ - описание переменной, для всех i=1,...,m v_i - переменная, t_i - терм. Терм t_1 не содержит свободных переменных. Для i=2,..., m свободными переменными терма t_i могут быть только переменные $v_1, v_2,...,v_{i-1}$. Все переменные $v_1, v_2,..., v_m$ попарно различны. Префикс может отсутствовать. Тело правила может быть одного из следующих видов:

(1) <префикс> P $\rightarrow$ $Op1(AF_1)$ & $Op1(AF_2)$ & … & $Op1(AF_{x1})$ & $Op2(F_1,t'_1)$ & $Op2(F_2,t'_2)$ & … & $Op2(F_{x2},t'_{x2})$ & $Op3(I_1,t''_1)$ & $Op3(I_2,t''_2)$ & … & $Op3(I_{x3},t''_{x3})$ & S_1, …,

S_k, где P – условие правила, задаваемое формулой, $Op1(AF_1)$ & $Op1(AF_2)$ & … & $Op1(AF_{x1})$ & $Op2(F_1,t'_1)$ & $Op2(F_2,t'_2)$ & … & $Op2(F_{x2},t'_{x2})$ & $Op3(I_1,t''_1)$ & $Op3(I_2,t''_2)$ & … & $Op3(I_{x3},t''_{x3})$ & S_1, …, S_k - следствие правила, где $Op1(AF_1)$ & $Op1(AF_2)$ & … & $Op1(AF_{x1})$ – операторы формирования кортежей, определяемые формулами AF_1, …, AF_{x1}, $Op2(F_1,t'_1)$ & $Op2(F_2,t'_2)$ & … & $Op2(F_{x2},t'_{x2})$ – операторы формирования кортежей, определяемые функциональными термами F_1, …, F_{x2} и значениями, представленными термами t'_1, …, t'_{x2}, соответственно, $Op3(I_1,t''_1)$ & $Op3(I_2,t''_2)$ & … & $Op3(I_{x3},t''_{x3})$ – операторы исключения вариантов, определяемые предметными именами I_1, …, I_{x3} и значениями, представленными термами t''_1, …, t''_{x3}, соответственно, S_1, …, S_k - ограниченные кванторные операторы вида (& (v : t) f), причем f является конъюнкцией операторов, вид которых определен выше.

(2) <префикс> P → $Mod(S_1, …, S_k)$, где P – формула, $Mod(S_1, …, S_k)$ – вызов модуля с именем Mod, которому передаются параметры – значения S_1, …, S_k; каждое S_i является термом, значение которого есть имя.

Схема правила имеет вид:

(3) NameSch([.КОНСТ]w_1, [.КОНСТ]w_2, …, [.КОНСТ]w_n): rule, где

NameSch – имя схемы, w_1, w_2, …, w_n – формальные параметры схемы, rule – тело схемы (правило вида 1 или 2). Запись «[.КОНСТ]» означает, что «.КОНСТ» может отсутствовать. Отсутствие .КОНСТ означает, что соответствующим фактическим параметром может быть некоторое имя; в противном случае фактическим параметром может быть константа. Все переменные–параметры схемы должны использоваться в теле схемы.

Конкретизация схемы имеет вид:

(4) NameSch(zw_1, zw_2, …, zw_n), где NameSch – имя схемы, zw_1, zw_2, …, zw_n – термы - фактические параметры схемы. Схема правила является аналогом описания подпрограммы, а конкретизация схемы – аналогом вызова подпрограммы.

Логическим модулем называется конструкция вида <ModName, CallModsName, GlobD, LocD, R>, где ModName – имя модуля, CallModsName – множество имен вызываемых модулей, GlobD – множество описаний глобальных имен, LocD - множество описаний локальных имен, R – множество правил, схем и конкретизаций схем.

Процесс логического вывода (ПЛВ) определяется как исчисление со входом. Рабочая среда модуля характеризуется множеством состояний. Состояние рабочей среды модуля δ_i есть множество пар вида <n, $value_i(n)$>, где n – имя, $value_i(n)$ – значение, сопоставленное этому имени в состоянии δ_i. Начальное состояние δ_1 формируется в результате ввода исходных данных. Каждое следующее состояние получается из предыдущего в результате применения некоторого правила, конкретизации схемы или вызова модуля.

Для всех имен n, определенность значения которых есть "точное", выполнено: $value_i(n) = value_j(n) = value_1(n)$ для любых i и j. Для всех предметных имен, определенность значения которых "неточное", выполнено: $value_{i+1}(n) \subseteq value_i(n) \subseteq value_1(n)$ для любого i. Сопоставленное такому имени n неточное значение $value_i(n)$ есть множество вариантов $\{c_1,c_2,…,c_m\}$ (неточное значение), где c_i ($1 \le i \le m$) – значение, принадлежащее множеству, задаваемому сортом имени n. В процессе логического вывода из этого множества исключаются варианты. Если при вводе исходных данных имени n не сопоставлено значение, то будем считать, что это значение есть Ω ("неопределенное значение"). Такое значение не может изменяться в процессе логического вывода.

Для всех функциональных и предикатных имен, определенность значения которых "неточное", выполнено: $value_i(n) \subseteq value_{i+1}(n)$ для любого i. Сопоставленное имени n неточное значение $value_i(n)$ есть множество кортежей $\{(c_1,c_2,\ldots,c_m:c_0)\}$, где c_i $(1 \le i \le m)$ – значение, принадлежащее множеству, задаваемому сортом аргумента функции или предиката, указанному при описании имени n; для функционального имени c_0 - значение, принадлежащее множеству, задаваемому сортом результата функции; для предикатного имени $c_0 \in$ {истина, ложь}. Из приведенного соотношения $value_i(n) \subseteq value_{i+1}(n)$ следует, что на каждом шаге логического вывода к множеству кортежей $value_i(n)$ могут быть добавлены новые кортежи. При вводе исходных данных имени n может быть сопоставлено пустое множество кортежей. Будем обозначать $arg_i(n) = \{(c_1,c_2,\ldots,c_m) \mid (c_1,c_2,\ldots,c_m:c_0) \in value_i(n)\}$.

Обозначим ξ - подстановку вариантов вместо предметных имен, входящих в терм или формулу, $\theta = \{v_1/c_1, \ldots, v_m/c_m\}$ – подстановка значений вместе переменных, $J_{\delta i}((t, \xi), \theta)$ – значение терма или формулы t при замене предметных имен значениями, задаваемыми подстановкой ξ, а переменных – значениями, задаваемыми подстановкой θ. Обозначим Ξ_i – множество подстановок ξ, возможных при состоянии δi. Если терм или формула t содержит предметные имена, определенность значения которых "неточное", то множество $\{J_{\delta i}((t, \xi), \theta) \mid \xi \in \Xi i\}$ содержит более одного элемента. Оно задает множество вариантов значений терма или формулы. Зафиксируем имя n и вариант var $\in$ valuei(n). Обозначим $\Xi_i(n,var)$ – подмножество подстановок ξ, возможных при состоянии δ_i; имени n во всех подстановках сопоставлен вариант var.

Правило вида (1) $(v_1:t_1)(v_2:t_2)...(v_m:t_m)$ P $\rightarrow$ $Op1(AF_1)$ & $Op1(AF_2)$ & … & $Op1(AF_{x1})$ & $Op2(F_1,t'_1)$ & $Op2(F_2,t'_2)$ & … & $Op2(F_{x2},t'_{x2})$ & $Op3(I_1,t''_1)$ & $Op3(I_2,t''_2)$ & … & $Op3(I_{x3},t''_{x3})$ & S_1, …, S_k применимо при текущем состоянии δ_i рабочей среды, если выполнены следующие условия: для всех j, $1 \le j \le m$ множество $\{J_{\delta i}((t_j, \xi), \theta) \mid \xi \in \Xi_i\}$ состоит из одного элемента; существует подстановка значений вместо переменных, определяемая префиксом правила и состоянием δ_i, при которой истинно условие правила; могут быть произведены изменения состояния рабочей среды.

В результате применения правила состояние рабочей среды изменяется в соответствии с семантикой формул $Op1(AF_1)$ & $Op1(AF_2)$ & … & $Op1(AF_{x1})$ & $Op2(F_1,t'_1)$ & $Op2(F_2,t'_2)$ & … & $Op2(F_{x2},t'_{x2})$ & $Op3(I_1,t''_1)$ & $Op3(I_2,t''_2)$ & … & $Op3(I_{x3},t''_{x3})$ & S_1, …, S_k. Опишем правила изменения состояния рабочей среды в зависимости от вида компонентов следствия.

Рассмотрим $AF_j = p(t_1,...,t_k)$, причем при описании предикатного имени p определенность значения C = неточное. Изменения состояния рабочей среды могут быть произведены при подстановке θ, если $J_{\delta i}(p(t_1,...,t_k), \theta) = \Omega$. В этом случае $value_{i+1}(p) = value_i(p) \cup \{(J_{\delta i}((t_1,...,t_k), \theta)$: true)}, т.е. к множеству кортежей отношения с именем p добавляется кортеж, полученный из аргументов атомной формулы в результате выполнения подстановки θ, причем отношение с именем p на этом кортеже истинно. Будем говорить, что при подстановке θ может быть совершен переход в противоречивое состояние, если $J_{\delta i}(p(t_1,...,t_k), \theta)$ = false. В этом случае выполнение модуля завершается по противоречию.

Рассмотрим $AF_j = \neg\, p(t_1,...,t_k)$, причем при описании предикатного имени p определенность значения C = неточное. Изменения состояния рабочей среды могут быть произведены при подстановке θ, если $J_{\delta i}(p(t_1,...,t_k), \theta) = \Omega$. В этом случае $value_{i+1}(p) = value_i(p) \cup \{(J_{\delta i}((t_1,...,t_k), \theta)$: false)}, т.е. к множеству кортежей отношения с именем p

добавляется кортеж, полученный из аргументов атомной формулы в результате выполнения подстановки θ, причем отношение с именем p на этом кортеже ложно. Будем говорить, что при подстановке θ может быть совершен переход в противоречивое состояние, если $J_{\delta i}(p(t_1,...,t_k), \theta) = true$. В этом случае выполнение модуля завершается по противоречию.

Пусть $Op_j(F_j,t'_j)$ имеет вид $f(t_1,...,t_k) = t'_j$, причем при описании функционального имени f определенность значения C = неточное. Изменения состояния рабочей среды могут быть произведены при подстановке θ, если $J_{\delta i}(f(t_1,...,t_k) = t'_j), \theta) = \Omega$. В этом случае $value_{i+1}(f) = value_i(f) \cup \{(J_{\delta i}((t_1,...,t_k), \theta): J_{\delta i}(t'_j, \theta)\}$, т.е. к множеству кортежей функции с именем f добавляется кортеж, который образуют значения аргументов функционального терма, полученные в результате выполнения подстановки θ, и соответствующее этому кортежу значение функции с именем f, полученное из терма t'_j в результате выполнения подстановки θ. Будем говорить, что при подстановке θ может быть совершен переход в противоречивое состояние, если $J_{\delta i}(f(t_1,...,t_k), \theta) \neq J_{\delta i}(t'_j, \theta)$. В этом случае выполнение модуля завершается по противоречию.

Рассмотрим оператор исключения вариантов $Op3(I_j,t''_j)$, в котором использовано предметное имя I_j, при описании которого определенность значения C = неточное. Изменения состояния рабочей среды могут быть произведены при подстановке θ, если вариант $J_{\delta i}(t''_j, \theta) \in value_i(I_j)$. В этом случае $value_{i+1}(I_j) = value_i(I_j) \setminus \{J_{\delta i}(t''_j, \theta)\}$, т.е. в результате из множества вариантов, сопоставленных имени I_j в состоянии δ_i, исключен тот вариант, который задан термом t''_j. Будем говорить, что при подстановке θ может быть совершен переход в противоречивое состояние, если $value_i(I_j) \setminus \{J_{\delta i}(t''_j, \theta)\} = \varnothing$. В этом случае выполнение модуля завершается по противоречию.

Рассмотрим S_j = (& (v: t) f), где формула f не содержит операторов исключения вариантов. Тогда для каждого значения переменной v, принадлежащего множеству $(J_{\delta i}(t),\theta)$, состояние рабочей среды модуля изменяется в соответствии с семантикой операторов формирования кортежей.

Рассмотрим S_j = (& (v: t) f), где формула f содержит операторы исключения вариантов. Тогда для каждого значения переменной v, принадлежащего множеству $(J_{\delta i}(t),\theta)$, состояние рабочей среды модуля изменяется в соответствии с семантикой выполнения таких операторов.

Правило вида (2) применимо при текущем состоянии δ_i рабочей среды, если выполнены следующие условия: для всех j, $1 \le j \le m$ множество $\{J_{\delta i}((t_j, \xi), \theta) \mid \xi \in \Xi_i\}$ состоит из одного элемента; существует подстановка значений вместо переменных, определяемая префиксом правила и состоянием δ_i, при которой истинно условие правила. В результате выполнения правила происходит вызов модуля Mod, которому передаются значения, сопоставленные именам S_1, …, S_k. Эти значения входят в состояние δ_1 для вызываемого модуля. В результате выполнения модуля значения, сопоставленные именам S_1, …, S_k в состоянии δ_{i+1} вызывающего модуля будут совпадать со значениями, сопоставленными этим именам в заключительном состоянии для модуля Mod.

Процесс логического вывода завершается, если достигнуто такое состояние δ_E (заключительное состояние), в котором не применимо ни одно из правил.

Правило вывода для конкретизации схемы определяется видом схемы. При этом формальным параметрам схемы сопоставляются значения фактических параметров, формируемые при текущем состоянии ПЛВ.

Теорема 1 (о конфлюэнтности). Пусть S, S', S'' – состояния процесса логического вывода. Если существуют выводы $S \Rightarrow S'$ и $S \Rightarrow S''$, то существует непротиворечивое состояние ПЛВ S''' такое, что $S \Rightarrow S' \Rightarrow S'''$ и $S \Rightarrow S'' \Rightarrow S'''$.

Теорема 2 (о конфлюэнтности при наличии противоречия). Пусть S_p - противоречивое состояние ПЛВ. Тогда если существуют выводы $S \Rightarrow S' \Rightarrow S_p$ и $S \Rightarrow S''$, то существует вывод $S \Rightarrow S'' \Rightarrow S_p$.

Глава 3 диссертации содержит описание схем распараллеливания процесса логического вывода.

В работе для построения параллельной системы продукций предлагается использовать архитектуру, когда отдельному процессу предлагается роль диспетчера (управляющий процесс), остальным процессам – роль обрабатывающего процесса (зависимые процессы). Управляющий процесс производит ввод-вывод данных и их синхронизацию, а также обмен данными с зависимыми процессами; он определяет очередность выполнения правил и назначает каждому процессу подпрограмму для обработки правила. Каждый зависимый процесс выполняет подпрограмму, реализующую процесс логического вывода для правила, то есть производит поиск всех подстановок, на которых истинно условие применимости правила, и для каждой такой подстановки выполняет действия, определяемые следствием правила, а полученные данные передает в другие процессы.

В работе показано, что естественная схема распараллеливания, когда каждому правилу назначается отдельный процесс, является неэффективной, поскольку в этом случае производится много лишних действий, требуемых для отслеживания появления новых данных для правил. В работе определены 7 схем распараллеливания процесса логического вывода: на основе множества активных правил, на основе свойств информационного графа, для правил с префиксом, путем разбиения области значений объектов внутри правила, распараллеливание модулей, вызываемых из одного модуля, а также распараллеливание модульной программ. Схемы разбиты на две группы: в схемах первой группы для распараллеливания вычислений используются связи между правилами, отображаемыми в формируемом при компиляции информационном графе (ИГ) модульной программы, а схемы второй группы используются при распараллеливании вычислений внутри отдельного правила; в них учитываются свойства этого правила.

Рассмотрим схему распараллеливания процесса логического вывода на основе свойств информационного графа.

Управляющий процесс:

НАЧАЛО вычислений: $\mu(P_f)$ = число доступных процессов; $P_w = \varnothing$. Блок 1.

ЦИКЛ: Пока есть iCurParentsAr[i] = 0 или $P_w \neq \varnothing$, делаем: Блок 2; Блок 3. Конец ЦИКЛА. Блок 4.

КОНЕЦ вычислений.

Здесь P_f – множество свободных процессов, P_w – множество занятых процессов, $\mu(P)$ – число элементов множества P.

Блок 1 (*Запускаем все правила, соответствующие корневым вершинам*):

Для каждого свободного процесса j из P_f делаем:

Для каждого элемента i массива iCurParentsAr: Если iCurParentsAr[i] = 0,

то Send(Q_i, i, j); iCurParentsAr[i] = -1; $P_f = P_f \setminus \{j\}$; $P_w = P_w \cup \{j\}$.

Конец блока 1.

Здесь iCurParentsAr – копия iParentsAr, при этом если в ИГ есть циклы, то значения элементов массива iCurParentsAr такие: если iLoopRootAr[i] = 2, то

iCurParentsAr[i] = iLoopAr[i]. iLoopRootAr – массив целых чисел размерности $\mu(\Pi_m)$, в котором элемент i массива равен 2, если правило i является входом в цикл, равен 1 – если правило i принадлежит циклу, 0 – не принадлежит циклу. iLoopAr – массив целых чисел размерности $\mu(\Pi_m)$, в котором элемент i массива равен -1, если правило i не принадлежит циклу, 0 - принадлежит циклу, и равен количеству прямых предков, не принадлежащих циклу, при условии, что правило i является входом в цикл. iParentsAr – массив целых чисел размерности $\mu(\Pi_m)$, в котором элемент i массива равен количеству прямых предков правила i.

Блок 2 (*Принимаем результирующие данные и синхронизируем их*):

1. Recv(Z, i, j); $P_w = P_w \setminus \{j\}$; $P_f = P_f \cup \{j\}$.
2. iCurParentsAr[i] = -2.
3. Для всех вершин k таких, что iCurParentsAr[k] > 0, делаем:

3.1. Если IncMatrix[i,k] = 1, то iCurParentsAr[k] = iCurParentsAr[k] – 1;

3.2. Если iLooptAr[i] > 0, и iCurParentsAr[k] = -2, и iLoopAr[k] > 0, и THEN(i) $\cap$ IF(k) $\neq \varnothing$, то iChangedLoopAr[k] = 1.

4. Data = Data $\cup$ Z;

Конец блока 2.

Recv(Z, i, j) принимает из процесса j набор данных Z, которые являются результатом вычисления правила i. iChangedLoopAr – массив целых чисел размерности $\mu(\Pi_m)$, в котором элемент i массива равен 1, если циклическое правило i было сначала вычислено, а потом для объектов, входящих в его условие, появились новые значения; иначе элемент i равен 0. IncMatrix – двумерный массив размерности $\mu(\Pi_m) \times \mu(\Pi_m)$, где элемент (i,j) матрицы равен 1, если j-тое правило является прямым потомком i-того правила, иначе элемент равен 0.

Блок 3 (*Загружаем свободные процессы правилами, готовыми к вычислению*):

Для каждого свободного процесса j делаем:
  Для каждого элемента i массива iCurParentsAr:
    Если iCurParentsAr[i] = 0,
    то Send(Q_i, i, j); iCurParentsAr[i] = -1.

Если $P_w = \varnothing$, и нет элементов k массива iCurParentsAr таких, что iCurParentsAr[k] = 0, то:
  Для каждого элемента t массива iCurParentsAr:
    Если iCurParentsAr[t] = -2 и iLoopAr[t] > 0, то:
      Для всех вершин s:
        Если LoopMatrix[t,s]=1 и iCurParentsAr[s]>0,
        То iCurParentsAr[s] = 0;
  Переход в Блок 3.

Конец блока 3.

Блок 4 (*Готовим правила, соответствующие циклическим вершинам, к повторным вычислениям*):

Если существуют элементы i массива iChangedLoopAr такие, что iChangedLoopAr[i] = 1, то:
  Для всех правил делаем:
    Если k – (дальний) потомок правила i,
    то iChangedLoopAr[k] = 1.
  Для каждого элемента t массива iChangedLoopAr:
    Если iChangedLoopAr[t] = 1,
    то iCurParentsAr[t] = iParentsAr[t];
    Если iLoopAr[t]>1, то iCurParentsAr[t]=0.
  Переход в Блок 3.
иначе:

Конец блока 4.

Здесь iChangedLoopAr – массив целых чисел, в котором элемент с номером i равен 1, если циклическое правило i было сначала вычислено, а потом для объектов, входящих в его условие, появились новые значения; иначе элемент с номером i равен 0. LoopMatrix - двумерный массив размерности $\mu(\Pi_m) \times \mu(\Pi_m)$, копия массива IncMatrix, в которой обнулены все вершины, не имеющие предков или потомков.

Зависимый процесс:	wCalc(i, Z, Q);
НАЧАЛО вычислений:	wSend(Q, i);
wRecv(Z, i);	КОНЕЦ вычислений

Здесь wRecv(Z, i) – процедура, которая принимает из управляющего процесса набор данных Z, который содержит значения объектов, входящих в условие правила i; wSend(Q, i) – процедура, которая пересылает в управляющий процесс набор данных Q, которые являются результатом вычисления правила i; wCalc(i, Z, Q) – процедура, вычисляющая правило i с помощью логического вывода.

Утверждение. Время выполнения программы T' при применении схемы 1 лежит в пределах: $(\Sigma t_i) / \mu(P) \le T' \le \Sigma t_i$, где Σt_i - сумма времен выполнения каждого правила, а $\mu(P)$ – максимальное число независимых ветвей ИГ.

В работе даны теоретические оценки времени выполнения программы при применении различных схем.

Глава 4 диссертации содержит описание алгоритма управления выбором схем распараллеливания логической программы, а также методов реализации и архитектуры системы параллельного программирования.

Рассмотрим ограничения, налагаемые архитектурой многопроцессорной системы и структурой ИГ. Пусть в модуле m количество правил равно $\mu(\Pi_m)$, для него определено максимальное количество процессов P_{opt}, которые можно выполнять параллельно друг с другом, при условии, что правила сначала вычисляются полностью и количество доступных рабочих процессов равно $\mu(P)$. В реальных задачах логический модуль может содержать сотни правил, а ветвистость его информационного графа может быть крайне малой. С другой стороны, реальные кластеры предоставляют не более нескольких десятков процессов. В этом случае $\mu(P) << \mu(\Pi_m)$, но так как в этом случае нельзя одновременно вычислять более P_{opt} правил, то будет правильней рассмотреть отношение чисел P_{opt} и $\mu(P)$. В случае, когда $P_{opt} < \mu(P)$, получаем простаивающие на протяжении всех вычислений рабочие процессы, с другой стороны, если $P_{opt} > \mu(P)$, можно в какие-то моменты времени задействовать все процессы. Также на использование процессов влияет строение графа, который может иметь разную ветвистость на разных уровнях.

Если использовать схему передачи кортежей между процессами, системе необходимо иметь в своем распоряжении количество рабочих процессов, равное количеству правил, в то время как реальные кластеры предоставляют не более нескольких десятков процессов. Вдобавок к этому возникают повышенные расходы на передачу кортежей между процессами и систему синхронизации данных.

В работе объединены оба способа распараллеливания правил, в результате чего система параллельного программирования использует максимальное количество доступных процессов в каждый момент времени в соответствии со схемами, описанными в главе 3. Для этого система производит вычисление всех готовых к выполнению правил из ИГ. Если после этого остались незадействованные рабочие процессы, система передает этим процессам вычисление правил-потомков, предки которых находятся в обработке, при этом происходит обмен вновь вычисленных кортежей между процессами, обрабатывающими связку правил «предок-потомок».

Ниже приводится описание алгоритма работы управляющего процесса по выбору схем распараллеливания правил. Для модуля m построен информационный граф, в котором количество правил $\mu(\Pi_m)$, количество свободных рабочих процессов $\mu(P)$. Следует отметить, что время выполнения каждой вершины-правила неизвестно. Следовательно, назначение готового к выполнению правила свободному рабочему процессу можно произвести только динамически в процессе логического вывода.

1. Находятся все готовые к выполнению правила-вершины ИГ. Если в следствии готового к выполнению правила находится вызов модуля, управляющий процесс передает управление на вычисление свободному рабочему процессу из множества $\mu(P)$. Если в следствии готового к выполнению правила нет вызова модуля, то оно передается на вычисление свободному рабочему процессу из множества $\mu(P)$. Если свободных процессов после этого не осталось, переход на шаг 4, иначе шаг 2.

2. Из ИГ выбираются правила-вершины такие, чтобы их единственным предком была вершина, которая находится в обработке. Каждое найденное правило передается на вычисление в свободный рабочий процесс, при этом данный процесс может получать результирующие кортежи от процесса, обрабатывающего правило-предок. Если свободных процессов после этого не осталось, переход на шаг 4, иначе шаг 3.

3. Если готовое к выполнению правило содержит префикс, то для каждого значения всех индексов префикса создается копия правила с подставленным значением индекса и пересылается в зависимый процесс. Все копии правила запускаются параллельно друг с другом. Если число свободных зависимых процессов меньше числа копий, то управляющий процесс принимает все результирующие кортежи и синхронизирует их с имеющимися у него данными, после чего запускает оставшиеся копии правил на освободившихся процессах.

4. Если какой-либо процесс закончил выполнение правила и выслал результаты, управляющий процесс принимает все результирующие кортежи и синхронизирует их с имеющимися у него данными. Если все правила выполнены и все рабочие процессы свободны, переход на шаг 5, иначе на шаг 1.

5. Если в ИГ существуют вершины, входящие в циклы, и в результате выполнения правил, соответствующих этим вершинам, появились новые значения объектов, которые входят в условия циклических правил, то строится подграф, в который входят все правила, являющиеся потомками данных циклических правил. ИГ заменяется на построенный подграф, и вновь производятся вычисления с шага 1. Если в ИГ циклов нет или при вычислении циклов новых значений не появилось, то переход на шаг 6.

6. Если модуль, правила которого были выполнены на предыдущих шагах, был вызван из другого модуля, то управляющий процесс передает управление в вызвавший его процесс. Переход на шаг 1. Если модуль является главным, то конец вычислений.

Зависимый процесс выполняет алгоритм 2.

1. Рабочий процесс принимает от управляющего номер правила и входные данные и начинает выполнение правила.

2. Если во время выполнения правила пришла команда на прием кортежей из другого процесса, то данный процесс принимает присланные кортежи и синхронизирует с имеющимися данными.

3. Если во время выполнения правила пришла команда пересылать вычисленные кортежи в другой процесс, то все вычисленные кортежи пересылаются в указанный процесс.

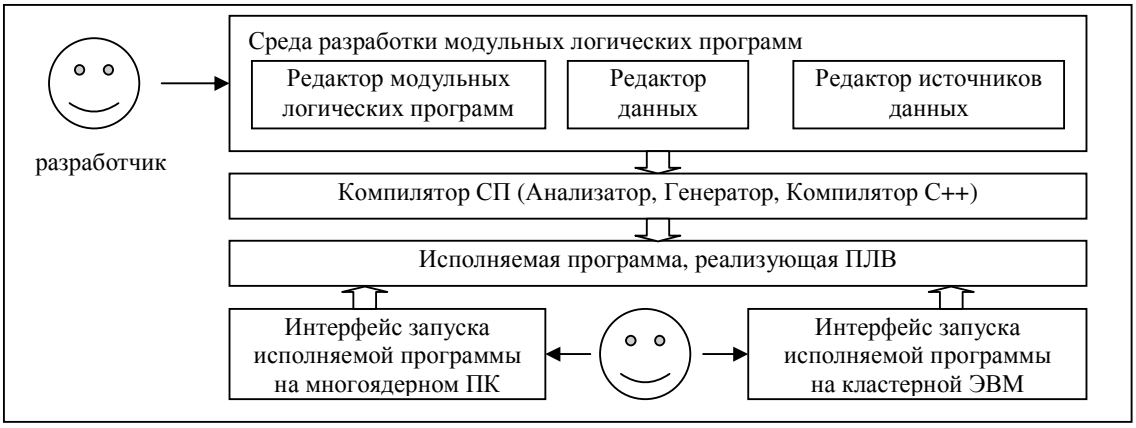


Рис. 1. Схема взаимодействия пользователей с СП (тонкая стрелка – взаимодействия пользователя с программой, толстая стрелка – взаимодействие между программами)

4. После выполнения правила все результирующие данные пересылаются в управляющий процесс.

Разработаны две версии системы продукций: первая версия создана для работы на отдельном многоядерном персональном компьютере (рис. 1), вторая версия для вычислений требует доступа к многопроцессорной кластерной ЭВМ.

Если исполняемая программа сгенерирована для использования на кластерной ЭВМ, то пользователю необходимо использовать интерфейс запуска программы на кластере.

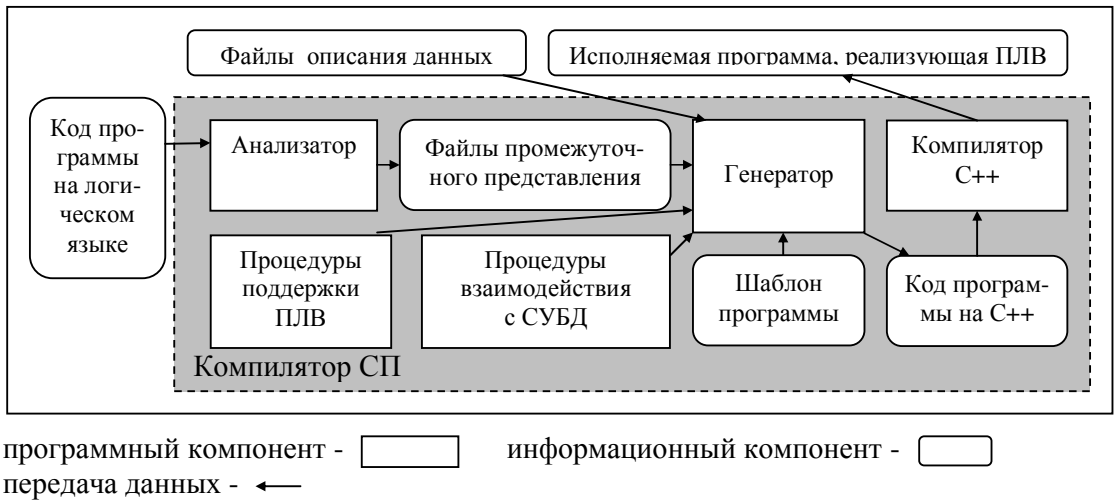


Рис. 2. Архитектурно-контекстная диаграмма компилятора СП

Система продукций состоит из набора компонентов, в ее состав входят среда разработки, анализатор, генератор (рис. 2), процедуры поддержки ПЛВ, процедуры взаимодействия с СУБД, шаблон генерируемой программы.

Пользователь системы продукций – разработчик прикладной программы, способный описать прикладную задачу на логическом продукционном языке. Пользователю в рамках среды разработки доступны редактор модульных логических программ, редактор данных, редактор описания данных, входящие в состав среды разработки. Результатом работы системы продукций является программное средство,

реализующее ПЛВ и решающее некоторую прикладную задачу. Пользователем этого средства является специалист той предметной области, которой принадлежит решаемая задача.

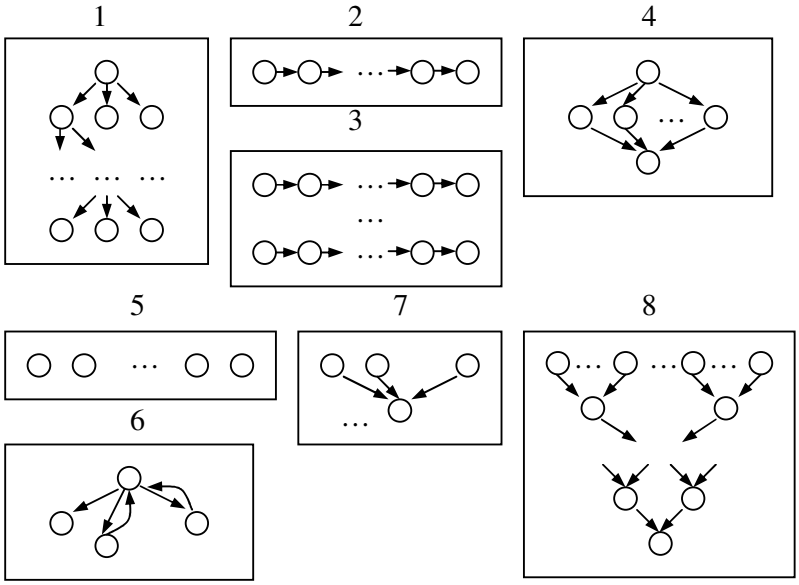


Рис. 3. Элементарные информационные графы

Глава 5 диссертации содержит результаты экспериментального исследования системы параллельного программирования.

Были проведены эксперименты, целью которых была проверка эффективности разработанных схем и методов реализации. В каждом эксперименте измерялось время работы сгенерированных программ для кластера на различных наборах данных.

Для экспериментов были выбраны логические программы, информационные графы которых являются элементарными, причем композиции таких программ дадут программу, имеющую информационный граф произвольного вида.

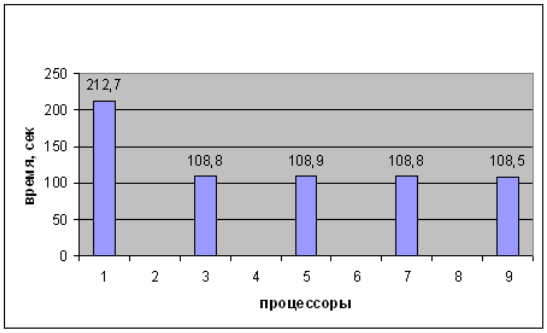
Было использовано девять элементарных информационных графов (рис. 3): (1) ИГ имеет одну начальную вершину и в каждую вершину (кроме начальной) входит одна дуга (рис. 3.1); (2) ИГ, в каждую вершину которого (за исключением начальной) входит единственная дуга, и из каждой вершины которого (за исключением конечной) входит единственная дуга, т.е. все вершины графа образуют цепочку, начинающуюся с начальной вершины и заканчивающуюся конечной (рис. 3.2); (3) ИГ содержит n независимых друг от друга цепочек вершин (рис. 3.3); (4) ИГ состоит из n вершин, причем единственная корневая вершина соединена с (n-2) потомками, каждый из которых соединен дугой с единственной конечной вершиной (рис. 3.4); (5) ИГ содержит n независимых друг от друга одиночных вершин и не содержит дуг (рис. 3.5); (6) ИГ состоит из n вершин, (n-1) из которых образуют цикл (рис. 3.6); (7) единственная вершина ИГ соответствует правилу, содержащему префикс; (8) ИГ состоит из n вершин, (n-1) из которых являются начальными, причем все начальные вершины соединены дугой с единственной конечной вершиной (рис. 3.7); (9) ИГ состоит из n вершин, k из которых являются начальными (k<n), и одна – конечной; каждая i-ая вершина (кроме начальных) имеет m_i (m_i <n) предков и одного потомка (рис. 3.8).

Все эксперименты разбиты на группы, номер которой соответствует номеру типа элементарного информационного графа в приведенном выше списке. Для каждого типа графа в одной группе экспериментов создавалась одна или несколько программ.

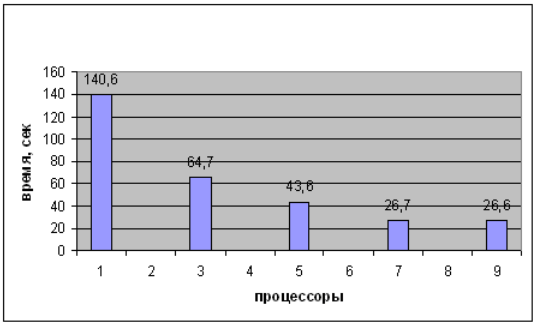
Программы отличаются количеством данных, передающихся между зависимыми между собой правилами. Таким образом, исследовалась зависимость времени работы программы от количества передаваемых данных между правилами и количества доступных процессоров.

Каждый эксперимент исполнялся несколько раз на одном и том же наборе входных данных, в описании результатов экспериментов приведено среднее время работы тестового приложения, которое равно сумме времен, деленных на количество тестовых прогонов. Следует отметить, что на скорость вычислений влияет время работы служебных процессов операционной системы, протокола MPI, задержки в коммуникациях. Особенно это проявляется в том случае, когда время выполнения правил маленькое (десятые доли секунды).

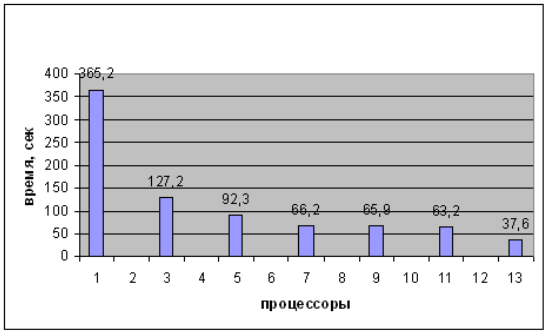
Для примера №1 (см. рис. 4) максимальное время выполнения программы, когда каждое правило выполнялось последовательно, составило 212,7 сек. при 1 рабочем процессе, минимальное – 108,5 сек. В параллельном режиме ускорение равно 1,96 при 9 рабочих процессах. Для примера №2 максимальное время работы составило 140,6 сек., минимальное – 26,6 сек. В параллельном режиме ускорение равно 5,28 при 9 рабочих процессах. Для примера №3 максимальное время работы составило 365,2 сек., минимальное – 37,6 сек. В параллельном режиме ускорение равно 9,71 при 13 рабочих процессах. Для примера №4 максимальное время работы составило 155,7 сек., минимальное – 52,8 сек. В параллельном режиме ускорение равно 2,94 при 9 рабочих процессах.



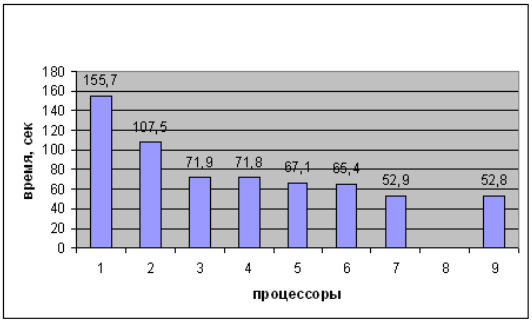
Пример №1



Пример №2



Пример №3



Пример №4

Рис. 4. Время выполнения программ при различных экспериментах

Кроме того, были проведены эксперименты на реальных задачах (медицины, вычислительной математики, химии), одним из которых был пример решателя одного из классов задач поиска путей синтеза химических соединений из предметной облас-

ти «химия». Приведем время работы примера по поиску путей синтеза химических соединений на различном числе процессоров:

Таблица 1. Время решение задачи поиска путей синтеза химических соединений

Число процессоров	1	2	4	6	9
модуль 1	32,1 сек	24,7 сек	12,9 сек	13,1 сек	13,1 сек

При последовательном выполнении (число рабочих процессоров равно 1) время работы составило 32,1 секунды. При параллельном режиме при увеличении числа процессоров время уменьшилось и при 6 процессорах составило 13,1 секунды. По сравнению с последовательным режимом ускорение равно 2,4 при 6 рабочих процессорах. Уменьшение времени работы связано с распараллеливанием вычислений с использованием схемы распараллеливания по информационному графу. Тем самым данные примеров подтверждают утверждения, приведенные в главе 3.

ОСНОВНЫЕ РЕЗУЛЬТАТЫ РАБОТЫ:

1. Разработана модель расширенного продукционного языка. Модель определяет средства для задания схем правил и их конкретизаций, содержит описание модулей и их вызовов, а также ограниченных кванторов. Язык допускает использование предметных, функциональных и предикатных символов, содержит теоретико-множественные операции и отношения, а также правила двух типов. Правила первого типа позволяют описывать действия, выполняемые над состояниями рабочей среды модуля, а правила второго типа – задавать условия вызова модулей правил. Модель языка определяет два типа правил вывода, позволяющих либо добавлять новые кортежи для объектов, имеющих функциональные или предикатные имена, либо исключать варианты для объектов с предметными именами.

2. На основе расширенной модели языка разработаны и исследованы схемы распараллеливания процесса логического вывода для системы конфлюэнтных продукций; описана организация процесса параллельного выполнения логического модуля, когда отдельному процессу предлагается роль диспетчера (управляющий процесс), остальным процессам – роль обрабатывающего процесса (зависимые процессы). Определение информационного графа обобщено на случай модульной логической программы. Предложены следующие методы распараллеливания: на основе множества активных правил, на основе свойств информационного графа, для правил с префиксом, путем разбиения области значений объектов внутри правила, распараллеливание модулей, вызываемых из одного модуля, а также распараллеливание модульной программы.

3. Проанализированы условия и ограничения, налагаемые структурой информационного графа логической программы и средой выполнения системы. С учетом этих ограничений разработан алгоритм управления выбором схем распараллеливания в процессе логического вывода.

4. Разработаны методы реализации системы параллельного программирования, а также ее архитектура. Система включает в себя такие компоненты, как среда разработки модульных логических программ, компилятор, набор процедур поддержки процесса логического вывода и процедур взаимодействия с СУБД. Описана схема

взаимодействия пользователей с системой, архитектурно-контекстные диаграммы компилятора и исполняемой программы.

5. В результате экспериментальных исследований свойств системы на различных наборах данных показана эффективность примененных схем распараллеливания процесса выполнения правил и алгоритма управления их выбором.

ОСНОВНЫЕ ПУБЛИКАЦИИ ПО ТЕМЕ ДИССЕРТАЦИИ

1. Артемьева И.Л., Тютюнник М.Б. Реализация прототипа конфлюэнтной системы продукций на многопроцессорной ЭВМ. // Высокопроизводительные вычисления на кластерных системах. Материалы второго межд. научно-пр. сем. / Под. ред. проф. Р.Г. Стронгина. Нижний Новгород: Изд-во Нижегородского государственного университета. - 2002. - С. 289-294.
2. Артемьева И.Л., Тютюнник М.Б. Методы распараллеливания вычислений для системы параллельного программирования на основе декларативных продукций. // II межд. конф. «Параллельные вычисления и задачи управления», Москва, 2004: сб. тр. [Электронный ресурс]. М: ИПУ РАН. - 2004. - С. 727-737.
3. Артемьева И.Л., Тютюнник М.Б. Прототип системы конфлюэнтных продукций для МВС-1000. // Информатика и системы управления. - 2004. - №2. - С. 133-143.
4. Тютюнник М.Б. Прототип продукционной системы параллельного программирования. // Тез. докл. Открытого Дальневосточного конкурса программных средств студентов, аспирантов и молодых специалистов. - 2004. - С. 45-49.
5. Артемьева И.Л., Тютюнник М.Б. Схемы распараллеливания вычислений для системы конфлюентных продукций. // Информатика и системы управления. - 2005. - №2. - С. 102-112.
6. Артемьева И.Л., Тютюнник М.Б. Анализ схем распараллеливания вычислений для системы конфлюэнтных продукций. // Научная сессия МИФИ-2006, сб. научн. тр. в 16 томах, т. 3. М: МИФИ. - 2006. - С. 128-129.
7. Артемьева И.Л., Тютюнник М.Б. Методы управления распараллеливанием вычислений в системе конфлюэнтных продукций. // Искусственный интеллект. НАНУ, Институт проблем искусственного интеллекта, Донецк. - 2006. - №4. – С. 107-117.
8. Артемьева И.Л., Тютюнник М.Б. Распараллеливание вычислений для системы конфлюэнтных продукций с использованием информационного графа. // III межд. конф. «Параллельные вычисления и задачи управления», Москва, 2006: сб. тр. [Электронный ресурс]. М.: ИПУ РАН. - 2006. - С. 1161-1171.
9. Тютюнник М.Б. Использование информационного графа при распараллеливании вычислений для системы конфлюэнтных продукций. // Информатика и системы управления. - 2006. - №1. - С. 181-192.
10. Артемьева И.Л., Тютюнник М.Б. Метод распараллеливания вычислений для модульных систем продукций. // Научная сессия МИФИ-2007, сб. научн. тр. в 17 томах, т. 3. М.: МИФИ. - 2007. - С. 118-119.
11. Артемьева И.Л., Тютюнник М.Б. Модель модульного языка декларативных конфлюэнтных продукций с ограниченными кванторами. // Искусственный интеллект. НАНУ, Институт проблем искусственного интеллекта, Донецк. - 2007. - №3. – С. 120-132.

12. Артемьева И.Л., Тютюнник М.Б. Модульная система конфлюэнтных продукций для многопроцессорной вычислительной системы. // Программные продукты и системы. - 2007. - №2. - С. 38-39.

13. Artemieva I.L., Tyutyunnik M.B. Parallelization of Logical inference for confluent rule-based system. // International Book Series «Information Science and Computing». Book 1 «Algorithmic and Mathematical Foundations of the Artificial Intelligence». - 2008. - PP.81-87.

14. Артемьева И.Л., Тютюнник М.Б. Методы управлением логического вывода для продукционной систем // Управление созданием и развитием систем, сетей и устройств телекоммуникаций. Тр. научн.-пр. конф., СПб.: НОЦ «Перспектива». - 2008. – С. 60-63.

15. Артемьева И.Л., Тютюнник М.Б. Управление распараллеливанием логического вывода для системы, основанной на правилах. // Научно-технические ведомости СПбГПУ. - 2008. - №3. - С. 99-103.

16. Артемьева И.Л., Тютюнник М.Б. Экспериментальное исследование системы параллельного программирования // Сб. научн. тр. научной сессии МИФИ-2008 в 15 томах. – Т. 10 «Интеллектуальные системы и технологии». М: МИФИ. - 2008 – С. 97-98.

17. Артемьева И.Л.. Тютюнник М.Б. Исследование распределенной системы логического программирования. // IV межд. конф. «Параллельные вычисления и задачи управления», Москва, 2008: сб. тр. [Электронный ресурс]. М.: ИПУ РАН. - 2008. - С. 1150-1160.

18. Artemieva I.L., Tyutyunnik M.B. Parallel Logical Inference for Confluent Rule-Based System. // First Russia and Pacific Conference on Computer Technology and Applications (RPC 2010): [Electronic res.]. - Vladivostok, Russia. - 2010. - PP. 171-176.

19. Артемьева И.Л., Никифорова Н.Ю., Тютюнник М.Б. Экспериментальные исследования свойств системы параллельного программирования. // V межд. конф. «Параллельные вычисления и задачи управления», Москва, 2010: сб. тр. [Электронный ресурс]. М: ИПУ РАН. - 2010. - С. 1157-1176.

20. Тютюнник М.Б. Система параллельного логического программирования РЕПРО-С. // Тез. докл. Открытого Дальневосточного конкурса программных средств студентов, аспирантов и молодых специалистов «Программист-2010». - 2010. - С. 20-23.

21. Артемьева И.Л., Тютюнник М. Б. Распараллеливающий компилятор для языка, основанного на правилах. Свидетельство о государственной регистрации программы для ЭВМ №2010614810.

Личный вклад автора. Все результаты, составляющие основное содержание диссертации, получены автором самостоятельно. В работах [11,12] автору принадлежит модель расширенного языка, в [2,5,6,8,10,13,18] – схемы распараллеливания процесса логического вывода, в [7,14,15] – методы управления выбором схем, в [1,3] – методы реализации параллельной системы продукций, в [16,17,19] – описание результатов экспериментального исследования системы, в [21] – реализация распараллеливающего компилятора.

Тютюнник Михаил Борисович

Разработка и исследование продукционной системы параллельного программирования

Автореферат

Подписано к печати 21.12.10 Усл. печ. л. 1,0 Уч.-изд. л. 0,8
Формат 60x84/16 Тираж 100 Заказ 41

Издано ИАПУ ДВО РАН. Владивосток, ул. Радио, 5
Отпечатано участком оперативной печати ИАПУ ДВО РАН
Владивосток, ул. Радио, 5