Многопроцессорные вычислительные системы
2005. №2(10)

УДК 004.43

© 2005 г. **И.Л. Артемьева**, канд. техн. наук,
М.Б. Тютюнник
(Институт автоматики и процессов управления ДВО РАН, Владивосток)

СХЕМЫ РАСПАРАЛЛЕЛИВАНИЯ ВЫЧИСЛЕНИЙ ДЛЯ СИСТЕМЫ КОНФЛЮЭНТНЫХ ПРОДУКЦИЙ[1]

В конфлюэнтных продукциях результат вывода не зависит от порядка применения правил, что позволяет рассматривать различные способы упорядочивания правил при организации процесса логического вывода и дает возможность распараллеливать этот процесс. В данной работе описано несколько схем распараллеливания вычислений для системы конфлюэнтных реляционных продукций и проведено их сравнение.

Введение

При разработке прикладных программ для многопроцессорных вычислительных комплексов используются либо специальные языки, имеющие средства задания параллельных процессов и организации их взаимодействия, либо традиционные языки программирования. В последнем случае задача распределения вычислений по процессам решается языковым процессором языка. Одним из классов языков, которые могут использоваться при создании прикладных программ, являются продукционные языки.

В системах конфлюэнтных продукций результат вычислений не зависит от порядка применения правил в процессе логического вывода. Это означает, что все правила независимы друг от друга, т.е. язык для записи конфлюэнтных продукций не требует никаких дополнительных языковых конструкций для написания параллельных программ. Задачу распределения вычислений по процессам решает языковой процессор продукционного языка – система конфлюэнтных продукций.

Естественной схемой распределения вычислений по процессам для системы конфлюэнтных продукций является схема, реализованная в прототипе системы программирования, разработанной в Институте автоматики и процессов управления ДВО РАН [1], где каждому процессу сопостав-

[1] Работа выполнена в рамках программы №17 фундаментальных исследований Президиума РАН «Параллельные вычисления и многопроцессорные вычислительные системы».

ляется правило системы продукций. Но такое распараллеливание, несмотря на то, что дает более эффективный способ вычислений по сравнению с однопроцессорной ЭВМ [2], не учитывает все возможности улучшения времени работы системы продукций.

Целью данной работы является анализ схем распараллеливания вычислений для систем конфлюэнтных продукций. Схемы разделены на два класса: глобальные и локальные. Глобальные– это схемы, учитывающие связи между правилами, а локальные – схемы, учитывающие связи внутри правил.

Система конфлюэнтных продукций

Правила (продукции) системы конфлюэнтных продукций могут быть следующих типов:

$$P(X) \rightarrow U_1(X_1) \,\&\, U_2(X_2)\&\ldots\&\, U_n(X_n), \qquad (1)$$

где $P(X)$ – логическое выражение-условие правила; $U_1(X_1)\,\&\ldots\&\,U_n(X_n)$ – следствие правила; $X_1, X_2,\ldots,X_n$ – векторы термов.

Каждая продукция должна удовлетворять условию на переменные:

$$(*)\ V(\{U_1(X_1), U_2(X_2),\ldots, U_n(X_n)\}) \subseteq V(P(X)),$$

где $V(O)$ – множество переменных, входящих в O (O может быть любой конструкцией). Условие правила является формулой. Элементом конъюнкции следствия может быть соотношение: атомная формула и обобщенная формула со знаком «&».

Правило с кванторами:

$$<K, \text{rule}>, \qquad (2)$$

где $K \equiv \{< io_1, Low_1, High_1>, \ldots, < io_{n(K)}, Low_{n(K)}, High_{n(K)}>\}$ либо $K \equiv \{< io_1, \Phi_1>, \ldots, < io_{n(K)}, \Phi_{n(K)}>\}$; причем каждая io_i – переменная, а каждый Low_i, $High_i$ – терм; каждое Φ_i – терм-множество либо функциональный терм; rule – правило вида (1), формулы которого содержат io_i, причем $io_i \neq io_j$ при $i \neq j$.

Переменную(ые) io_i будем называть индексом(ами) правила; Low_i, $High_i$ – нижней и верхней границами изменения индекса(ов) правила с кванторами (соответственно); множество Φ_i – областью изменения соответствующего индекса io_i обобщенного правила; правило rule – тело правила с кванторами – должно удовлетворять условию на переменные (*).

Определим типы термов.

1. Предметный символ d есть терм.
2. Переменная v есть терм.
3. Если $t^i_1, \ldots, t^i_m$ – термы, то $\{t^i_1, \ldots, t^i_{m(\Phi i)}\}$ – терм-множество.
4. Если $Zn \in \{+,*\}$; io – переменная; Low, High – термы; t – терм, содержащий io, то <Zn, (io: Low, High), t> – есть обобщенный терм. Здесь Zn – знак обобщенной операции; переменная io – индекс операции; Low, High – нижняя и верхняя границы изменения индекса операции соответственно; t – тело обобщенной операции.

5. Если Zn ∈ {+,*}; io – переменная; Ψ терм-множество либо функциональный терм; t – терм, содержащий io, то <Zn, (io: Ш), t> есть обобщенный терм. Здесь Zn – знак обобщенной операции; переменная io – индекс операции; множество Ш задает область возможных значений индекса операции; t – тело обобщенной операции.

6. Если tt – вектор термов, составленный из термов $t_1,\ldots t_{nt}$, f – функциональный символ, тогда f(tt) – терм. Такой терм будем называть функциональным термом.

7. Если tt – вектор термов, составленный из термов $t_1,\ldots t_{nt}$, v – переменная, тогда f(tt) – терм.

8. Если t_1 – терм, t_2 – терм, то t_1¤ t_2, (t_1¤ t_2) также термы, причем символ «¤» суть знак операции из множества {+, –, *, /}.

Определим типы формул:

1. Если tt – вектор термов, составленный из термов $t_1,\ldots t_{nt}$, p – предикатный символ, тогда p(tt) – формула. Такую формулу будем называть атомной формулой.

2. Если tt – вектор термов, составленный из термов $t_1,\ldots t_{nt}$, v – переменная, тогда v(tt) – формула.

3. Если t_1 – терм, t_2 – терм, то t_1@ t_2 – формула, где символ @ обозначает один из знаков соотношения =, >, <, ≠, ≥, ≤. Будем называть такую формулу соотношением.

4. Если Zn ∈ {&,∨}; io – переменная; Low, High – термы; f – формула, содержащая io, то <Zn, (io: Low, High), f> – обобщенная формула. Здесь Zn – знак обобщенной операции; переменная io – индекс операции; Low, High – нижняя и верхняя границы изменения индекса операции соответственно; f – тело обобщенной операции.

5. Если Zn ∈ {&,∨}; io – переменная Ψ терм-множество либо функциональный терм, f – формула, содержащая io, то <Zn, (io:Ш), f> есть обобщенная формула. Здесь Zn – знак обобщенной операции; переменная io – индекс операции; множество Ш задает область возможных значений индекса операции; f – тело обобщенной операции.

6. Если f_1 – формула, f_2 – формула, то f_1 ¤ f_2, (f_1 ¤ f_2) – формулы, причем символ «¤»∈ {&,∨}.

Классы схем распараллеливания вычислений

Процесс логического вывода (ПЛВ), реализованный на однопроцессорном компьютере, обрабатывает каждое правило логической программы последовательно, одно за другим. Многопроцессорная машина дает возможность построить работу программной системы, реализующей ПЛВ, как набор процессов, выполняемых параллельно разными процессорами компьютера. Одному из процессов предложена роль диспетчера, управляющего работой по подготовке данных и синхронизации полученных результа-

тов. Остальные процессы выполняют обработку каждого правила в соответствии с действиями, определяемыми схемами распараллеливания.

Введем некоторые понятия, используемые далее в описании схем.

Определение 1. Множеством активных правил (МАП) будем называть набор правил, которые система может обработать на текущем шаге процесса логического вывода. Правила, входящие в МАП, имеют как минимум один кортеж для каждого объекта (отношения или функции), входящего в условие правила, и могут выполняться параллельно и независимо друг от друга. На каждом следующем шаге процесса логического вывода правило попадает в МАП в том случае, если появился новый кортеж для любого из объектов, входящих в условие правила.

Определение 2. Информационным графом (ИГ) программы будем называть ориентированный граф G = <K, U>, где K – множество правил, а U – множество дуг, причем $U = \{(П_1, П_2) \mid П_1 \in K, П_2 \in K, THEN(П_1) \cap IF(П_2) \neq \varnothing\}$, здесь IF(П) – множество предикатных и функциональных символов условия правила, THEN(П) – множество предикатных и функциональных символов следствия правила.

В данной работе рассмотрены два класса схем распараллеливания вычислений. В глобальных схемах используются связи между правилами или группами правил, а в локальных – связи между данными внутри отдельного правила. Примерами глобальных схем являются схемы распараллеливания вычислений с использованием МАП и ИГ, а примерами локальных – схема распараллеливания вычислений для правил, использующих кванторы, и схема распараллеливания вычислений при поиске новых значений объектов внутри правила.

Схема распараллеливания вычислений с использованием множества активных правил

Введем обозначения. Пусть П’ – множество правил, выполняемых зависимыми процессами; П’’ – множество правил, выполненных зависимыми процессами; π – правило логической программы; СтартПроцесса(π) – набор действий по запуску процесса для вычисления правила π; ПолучитьРезультат(π, M) – набор действий по приему вычисленных данных M правила π из отработавшего процесса; Синхронизировать (M) – набор действий по синхронизации данных, полученных из отработавшего процесса, с данными, хранящимися в управляющем процессе; Вычислить(π, M) – набор действий по вычислению правила π, то есть по поиску новых значений M объектов, входящих вследствие правила.

С помощью введенных обозначений опишем процесс распараллеленного логического вывода.

Начало главного процесса; Сформировать МАП; П’ = ∅; П’’ = ∅;
Пока МАП ≠ ∅ выполнить: выбрать $\pi \in$ МАП; МАП = МАП \ {π};

СтартПроцесса(π); П’ = П’ ∪ {π}; Конец_цикла;
Пока ¬(МАП = ∅ и П’ = ∅) выполнить:
Если П’’ ≠ ∅, то ПолучитьРезультат(π’’, М); Синхронизировать (М); Сформировать МАП; П’’ = П’’ \ {π’’}; конец_условия;
Пока МАП ≠ ∅ выполнить: выбрать π ∈ МАП; СтартПроцесса(π);
МАП = МАП \ {π}; П’ = П’ ∪ {π}; конец_цикла;
конец_цикла; Конец главного процесса.

Зависимый процесс начинает обрабатывать правило при выполнении команды СтартПроцесса(π).

Начало зависимого процесса;
Вычислить(π, М); П’’ = П’’ ∪ {π}; П’ = П’ \ {π};
Конец зависимого процесса.

Прокомментируем приведенные алгоритмы. Вначале управляющим процессом формируется МАП. Оно содержит те правила, для объектов которых в исходных данных заданы значения и которые не зависят по данным от других правил. Далее в цикле поочередно каждому правилу из МАП сопоставляется зависимый процесс. Управляющий процесс ждет результата выполнения хотя бы одного из зависимых процессов, после чего происходит обновление МАП: это множество пополняется теми правилами, для объектов которых появляются новые означивания. В случае пополнения МАП новыми правилами происходит передача этих правил свободным процессам для вычислений. При формировании МАП очередь отработавших процессов увеличивается, однако они не могут получить новых данных для обработки до тех пор, пока не передадут результаты в управляющий процесс. Программа завершает работу тогда, когда не останется зависимых процессов, обрабатывающих правила, и МАП будет пустым. В данной схеме дополнительные расходы связаны с организацией поддержки МАП. Схема применена в прототипе системы [1], ее исследования приведены в [2].

Схемы распараллеливания вычислений с использованием информационного графа

В отличие от предыдущей схемы, где на каждом шаге процесса логического вывода строилось МАП, данная схема организует параллельное выполнение правил на основе анализа ИГ программы. Для этого на этапе трансляции формируется структура данных (описывающая ИГ), которая позволяет определить очередность выполнения правил на этапе исполнения.

Если программа не содержит зависимых друг от друга по данным правил, то ИГ не будет иметь циклов, а значит, на этапе трансляции программы можно определить такой порядок выполнения правил, при котором они будут выполняться не более одного раза.

Если программа содержит зависимые по данным правила, то ИГ будет иметь циклы. В этом случае циклу ИГ соответствует цикл правил. Каждый цикл выполняется до тех пор, пока правила из цикла изменяют состояние вывода.

Рассмотрим пример работы зависимых процессов.

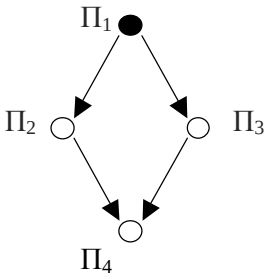

Рис. 1.1. Работа $П_1$.

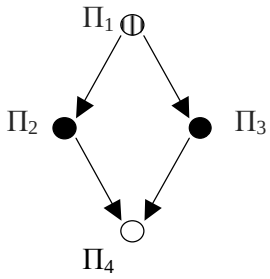

Рис. 1.2. Работа $П_2$ и $П_3$.

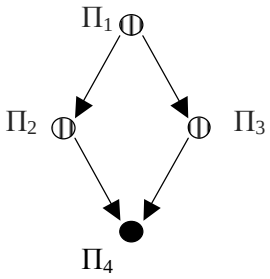

Рис. 1.3. Работа $П_3$.

Пусть в программе присутствуют четыре правила $П_1$, $П_2$, $П_3$, $П_4$, а ИГ программы имеет вид, приведенный на рис. 1. В программе $П_1$ – правило, которое будет выполняться первым; $П_2$ и $П_3$ – правила, зависимые по данным от $П_1$; $П_4$ – правило, зависимое по данным от правил $П_2$ и $П_3$. На рисунке кружками черного цвета обозначены правила, которые обрабатываются в текущий момент, кружками белого цвета – правила, которые еще не обрабатывались, а кружками в полоску – правила, которые уже отработали.

Управляющий процесс обходит ИГ и запускает процесс, соответствующий первому правилу $П_1$ (рис. 1.1). Ни одного процесса не может быть запущено до тех пор, пока не завершит работу $П_1$. Как только это правило завершило работу, могут быть запущены процессы, соответствующие $П_2$ и $П_3$ (рис. 1.2). И, наконец, происходит обработка правила $П_4$ сразу после завершения $П_2$ и $П_3$ (рис. 1.3).

Из примера видно, что при выполнении правила $П_1$ используется только один процесс, остальные простаивают. Параллельное выполнение правил происходит в том случае, когда граф разветвляется. Максимально возможное число процессов определяется числом ветвей в ИГ. Количество работающих параллельно процессов в этом случае определяется числом доступных процессов. Если их меньше числа ветвей, то управляющий процесс будет запускать только часть правил. Остальные будут запущены лишь по освобождении хотя бы одного процесса. Если в рассмотренном примере число доступных процессов равно трем, то число параллельно работающих правил не будет превышать двух вне зависимости от числа ветвей ИГ.

В следующей схеме также используется информационный граф. Данная схема предлагает выполнять очередное зависимое правило, не дожидаясь окончательной обработки текущего правила. В этом случае появ-

ление хотя бы одного кортежа в процессе выполнения правила дает старт на выполнение зависимого правила. Тогда максимально возможное число процессов будет равняться числу правил в программе. Но ограничением является число доступных процессов.

Рассмотрим пример работы управляющего и зависимых процессов для данной схемы при условии, что доступных процессов меньше, чем правил.

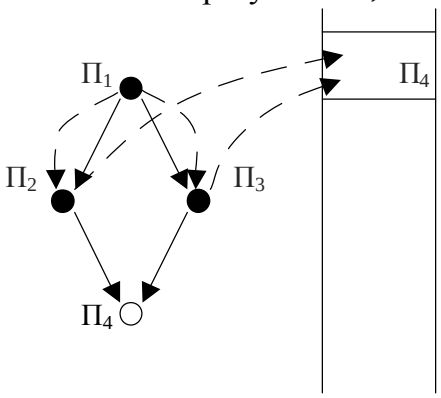

Рис. 2.1. Работа $П_1$, $П_2$ и $П_3$.

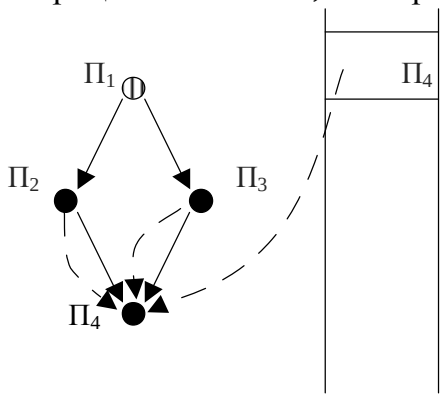

Рис. 2.2. Работа $П_2$, $П_3$ и $П_4$.

На рис. 2 показан такой же ИГ, как и на рис. 1. Пусть доступно четыре процесса, тогда один будет управляющим, а три – процессами для обработки правил. На рис. 2 вертикальная полоса справа от графа схематично обозначает управляющий процесс, который хранит и обрабатывает данные, принятые от зависимых процессов, обрабатывающих правила; стрелки с пунктирной линией – передача данных; кружки означают то же самое, что и на рис. 1.

Управляющий процесс сопоставляет трем процессам правила $П_1$, $П_2$, $П_3$. Правило $П_1$ все кортежи пересылает зависимым правилам $П_2$ и $П_3$, а те, в свою очередь, направляют результаты вычислений управляющему процессу.

Пусть $П_1$ закончило работу быстрее, чем $П_2$ и $П_3$, тогда управляющий процесс сопоставляет освободившемуся процессу правило $П_4$ (рисунок 2.2). Теперь $П_2$ и $П_3$ будут направлять результаты вычислений $П_4$; управляющий процесс также начнет пересылать $П_4$ скопившиеся кортежи.

Работа программы закончится, как только выполнятся все правила: $П_2$, $П_3$, $П_4$ и в управляющем процессе не останется скопившихся кортежей.

Если провести сравнение двух схем для приведенных примеров, то можно заметить, что в отличие от первой схемы, где работало в параллельном режиме максимум два процесса, во второй схеме задействованы все процессы, а это дает выигрыш во времени.

Схема распараллеливания вычислений для правил с кванторами

Тело правила с квантором должно выполняться столько раз, сколько разных значений имеет индекс этого правила, т.е. правило с кванторов оп-

ределяет цикл, телом которого является тело правила. Если индексов несколько, то получаем несколько вложенных циклов. Поэтому правила с кванторами легко поддаются распараллеливанию.

Рассмотрим пример правила с кванторами и объясним для него возможные схемы распараллеливания. Правило имеет вид:

<{<i, 3, 10>} a(i) & b(v) & v > 10 -> c(i+v) >.

Индексом является переменная i, нижняя граница задана числом 3, а верхняя – числом 10. Тело правила a(i) & b(v) & v > 10 -> c(i+v).

Первая схема распараллеливания предполагает следующие действия. Сначала надо «раскрыть» квантор, т.е. получить столько правил, сколько значений имеет индекс правила, при этом заменив вхождения переменной соответствующим значением. Далее полученный набор правил можно обрабатывать параллельно, так как правила не содержат зависимостей по данным между собой.

Применяя данную схему для примера, получим набор правил:

a(**3**) & b(v) & v > 10 -> c(**3**+v);
…
a(**10**) & b(v) & v > 10 -> c(**10**+v).

Здесь присутствует очевидная параллельность и независимость по данным, простейшая схема назначения правил процессам. Если такое правило является вершиной ИГ, то при применении схемы распараллеливания вычислений с использованием ИГ происходит «раскрытие» правила-узла, входящего в состав графа, в набор независимых друг от друга правил. В связи с тем, что «раскрытые» правила не имеют никаких связей по данным, их можно обрабатывать параллельно.

Отрицательным является большое количество правил, что требует большого количества процессов/процессоров; увеличивается время на синхронизацию и транспортировку данных между процессами.

Если границы изменения индекса заданы множеством и это множество известно на этапе трансляции, то число требуемых процессов известно до выполнения. Если диапазон изменения индекса зависит от исходных данных или результатов вычисления, то число требуемых процессов будет также зависеть от этих данных и результатов.

Рассмотрим вторую схему организации вычислений для правил с кванторами. Если множество значений квантора задается нижней и верхней границей, то можно внести этот диапазон значений вместе с переменной внутрь правила. Далее правило обрабатывается так же, как правило, не содержащее квантора.

Применяя вторую схему для примера, получим правило:

a(**i**) & b(v) & **i >2 & i < 11** & v > 10 -> c(**i**+v).

Схема распараллеливания вычислений при поиске новых значений объектов внутри правила

В данном случае процесс распараллеливания направлен на разделение области существующих значений для объектов, входящих в условие правила, и передачу каждой подобласти в соответствующий зависимый процесс, в котором и будет происходить поиск результата.

Пусть для правила имеем в условии **n** штук объектов **o**, для каждого $\mathbf{o_i}$ объекта существует множество кортежей значений аргументов **ar** в количестве **mi**.

$ar_1(o_1)$ $ar_1(o_2)\ldots$ $ar_1(o_n)$
$ar_2(o_1)$ $ar_2(o_2)\ldots$ $ar_2(o_n)$
… … …
$ar_{m1}(o_1)$ $ar_{m2}(o_2)\ldots$ $ar_{mn}(o_n)$

При стандартном переборе значений для поиска тех, которые удовлетворяют условию правила, происходит выбор и проверка условия среди множества **Args** всех значений:

$ar_1(o_1)$ $ar_1(o_2)$ … $ar_1(o_n)$,
$ar_1(o_1)$ $ar_1(o_2)$ … $ar_2(o_n)$,
$ar_1(o_1)$ $ar_1(o_2)$ … $ar_3(o_n)$,
… … …
$ar_1(o_1)$ $ar_2(o_2)$ … $ar_1(o_n)$,
$ar_1(o_1)$ $ar_2(o_2)$ … $ar_2(o_n)$,
$ar_1(o_1)$ $ar_2(o_2)$ … $ar_3(o_n)$
и т.д.

Очевидно, каждую строку значений можно проверять на совместимость с условием правила в параллельном режиме, т.е. одновременно. Таким образом, мы можем разбить множество значений объектов на непересекающиеся подмножества:

$Args = Arg_1 \cup Arg_2 \cup \ldots \cup Arg_k$.

$Arg_1 \cap Arg_2 \cap \ldots \cap Arg_k = \varnothing$.

Пример. $П_1$: a(v1, v2) & b(v3, v4) & условие(v1) & условие(v2) & условие(v3) & условие(v4) -> c(v1+v3).

Для каждого объекта правила $П_1$ имеется набор значений-кортежей:

Объект «a»	
Арг. 1	Арг. 2
a1[1]	a2[1]
a1[2]	a2[2]
…	…
a1[n]	a2[n]

Объект «b»	
Арг. 1	Арг. 2
b1[1]	b2[1]
b1[2]	b2[2]
…	…
b1[m]	b2[m]

Схема поиска всех кортежей объектов, удовлетворяющих условию правила $П_1$, выглядит в общем случае следующим образом (для обрабатывающего процесса):

```
Начало;
В цикле i от 1 до n выполнить:
  если условие(a1[i]) = «истина» и условие(a2[i]) = «истина», то
    В цикле j от 1 до m выполнить:
      если условие(b1[j]) = «истина» и условие(b2[j]) = «истина», то
        ВыполнитьСледствие(a1[i], a2[i], b1[j], b2[j]);
      конец_условия;
    конец_цикла;
  конец_условия;
конец_цикла;
Конец;
```

Видно, что происходит обход всех кортежей объектов, это осуществляется за счет вложенных циклов. Для распараллеливания подобной обработки правил можно разбить пространство поиска кортежей для первого объекта по количеству свободных процессов. Другими словами, имея k процессов, доступных для использования в вычислении правила, пересылаем первому процессу с 1-го по (n/k)-й кортежи первого объекта, второму процессу – с (n/k + 1)-й по (2*n/k)-й кортежи первого объекта, k-му процессу – с ((n-1)*n/k + 1) по n-й соответственно.

Тогда для примера для t-го процесса ($1 \le t \le k$) получим:

```
Начало;
В цикле i от 1 до ((t-1)* n/k+1) выполнить:
  если условие(a1[i]) = «истина» и условие(a2[i]) = «истина», то
    В цикле j от 1 до m выполнить:
      если условие(b1[j]) = «истина» и условие(b2[j]) = «истина», то
        ВыполнитьСледствие(a1[i], a2[i], b1[j], b2[j]);
      конец_условия;
    конец_цикла;
  конец_условия;
конец_цикла;
Конец;
```

Управляющий процесс производит распределение кортежей первого объекта, пересылая в соответствующий процесс копию соответствующих кортежей.

Для этой схемы следует рассматривать отдельные правила, так как схема является локальной. Она предполагает разделение области просмотра кортежей на части и передачу этих частей для обработки отдельным процессам. В таком случае зависимостей по данным не предполагается.

Заключение

В работе приводятся схемы распараллеливания, позволяющие эффективно исполнять процесс логического вывода на кластерной системе.

Описанные схемы распараллеливания для данной системы используют тот факт, что все правила являются независимыми друг от друга, т.е. система продукций обладает естественным параллелизмом.

Схемы распараллеливания, рассмотренные выше, разбиты на два класса: глобальные и локальные. Первые учитывают связи между правилами по передаче данных; эти связи находятся динамически или отражаются в графе передачи данных. Если при учете связей между правилами может быть построена последовательность правил, тогда при организации работы на параллельной системе распараллеливается процесс выполнения одного правила, т.е. применяется одна из подходящих локальные схем. Если же в графе существует несколько ветвей, то распараллеливание состоит в параллельном выполнении веток. В дальнейшей работе предполагается исследовать все возможные схемы распараллеливания, получить оценки времени выполнения для каждой из схем и выбрать оптимальную схему для реализации системы конфлюэнтных продукций на многопроцессорной ЭВМ.

При организации работы зависимого процесса, соответствующего одному правилу, могут применяться оптимизации процесса выполнения правил [3,4], что ускорит работу всей параллельной системы.

ЛИТЕРАТУРА

1. *Артемьева И.Л., Тютюнник М.Б.* Прототип системы конфлюэнтных продукций для МВС-1000 // Информатика и системы управления. 2004. №2(8). С.133-144.
2. *Артемьева И.Л., Тютюнник М.Б.* Методы распараллеливания вычислений для системы параллельного программирования на основе декларативных продукций // Тр. II международ. конф. "Параллельные вычисления и задачи управления". 2004; М.: ИПУ РАН, 2004. С.727-737.
3. *Клещев А.С., Чернойван К.Г.* Исследование методов оптимизаций вывода в системах декларативных продукций // Теория и практика систем с базами знаний. Владивосток: ДВО РАН, 1994. С.36-55.
4. *Клещев А.С., Чернойван К.Г.* Ускорение вывода в декларативных продукциях на основе пошаговых оптимизаций // Науч.-техн. информация. 1998. № 7. С.23-32.

Статья представлена к публикации членом редколлегии А.С. Клещевым.