УДК: 004.43

МЕТОДЫ РАСПАРАЛЛЕЛИВАНИЯ ВЫЧИСЛЕНИЙ ДЛЯ СИСТЕМЫ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ НА ОСНОВЕ ДЕКЛАРАТИВНЫХ ПРОДУКЦИЙ[1]

И.Л. Артемьева
Институт автоматики и процессов управления ДВО РАН
Россия, 690041, г. Владивосток, ул. Радио, 5
E-mail: artemeva@iacp.dvo.ru

М.Б. Тютюнник
Институт автоматики и процессов управления ДВО РАН
Россия, 690041, г. Владивосток, ул. Радио, 5
E-mail: humanoid@scientist.com

Ключевые слова: системы продукций, распараллеливание логического вывода
Key words: rule-based systems, parallel inference

Данная работа является описанием системы параллельного программирования для многопроцессорной ЭВМ на основе конфлюэнтных реляционных продукций. В ней рассматривается метод распараллеливания вычислений, приводятся результаты экспериментов, целью которых было сравнение времени выполнения для последовательной и многопроцессорной систем. Анализируются другие схемы распараллеливания для систем конфлюентных продукций.

PARALLEL COMPUTING METHODS FOR PARALLEL PROGRAMMING SYSTEM BASED ON DECLARATIVE PRODUCTIONS / I.L. Artemieva (Institute for Automation and Control Processes, 5 Radio, Vladivostok, 690041, Russia, E-mail: artemeva@iacp.dvo.ru), M.B. Tyutyunnik (Institute for Automation and Control Processes, 5 Radio, Vladivostok, 690041, Russia, E-mail: humanoid@scientist.com). The paper describes a parallel programming system based on confluent relational productions with an implementation in a multi-processor environment. There is a description of a method of parallel computing for logical inference, experiments results and the conclusions about their productivity in a single-processor and multi-processor environment. At the end the analysis follows which describes other schemes of parallel calculations for the systems based on confluent relational productions.

1. Введение

При появлении многопроцессорных ЭВМ рассматривались различные подходы для получения систем программирования для них.

Во-первых, создавались новые алгоритмические языки для написания параллельных алгоритмов. В таких языках явно присутствуют средства для создания параллельных процессов, для синхронизации их работы и организации их

[1] Работа выполнена в рамках программы № 17 фундаментальных исследований «Параллельные вычисления на многопроцессорных вычислительных системах» Президиума РАН на 2004 год.

взаимодействия. Примерами таких языков могут служить ответвления языка C и других алгоритмических языков.

Во-вторых, были разработаны языки, позволяющие работать параллельно с данными, задаваемыми массивами. Для подобных языков рассматриваются схемы распараллеливания на уровне данных. Примером подобного языка является HPF (High Performance Fortran).

Наконец, следует сказать о логических языках, которые, в основном, базируются на известном языке Prolog. Среди первых систем, основанных на логическом языке, отметим японскую систему, разработанную в рамках создания компьютеров пятого поколения, которая базируется на высокопродуктивном языке логического параллельного программирования KL1. Хотя KL1 во многом основывается на Prolog и поддерживает режим параллельного вывода, он, тем не менее, ориентирован на конкретную компьютерную архитектуру, и поэтому не реализует в полной мере логику предикатов первого порядка.

Таким же ответвлением, как и KL1, является язык n-Prolog, для которого был разработан свой механизм вывода. Этот механизм позволяет автоматически и незаметно распознавать параллельные части программы Prolog, однако не позволяет полностью распараллелить процесс логического вывода.

Исходя из этого, можно сделать вывод о том, что в настоящее время нет языков, которые в полной мере реализуют полностью распараллеленный логический вывод.

Система продукций, рассматриваемая в данной работе, относится к классу конфлюэнтных реляционных продукций. В конфлюэнтных системах продукций результат вычислений не зависит от порядка применения правил в процессе логического вывода. Это означает, что все правила являются независимыми друг от друга, т.е. система продукций обладает естественным параллелизмом и не требует никаких дополнительных языковых конструкций для написания параллельных программ.

Разрабатываемая система предназначена для параллельного программирования на основе конфлюэнтных продукций и исполнения сгенерированной программы на многопроцессорной вычислительной машине.

2. Краткое описание среды

Мультикомпьютер, на котором реализована система параллельного программирования, представляет собой Linux-кластер, состоящий из 16 узлов. Для организации взаимодействия параллельных процессов используется протокол MPI (реализация LAM). Прототип системы построен с использованием подхода «управляющий процесс – зависимый процесс», то есть, применительно к программному средству, объекты и их значения будут храниться в памяти, используемой управляющим процессом, а для передачи данных процессам-обработчикам управляющий процесс снимает копию памяти, где находятся данные. Результаты же обработки объектов, полученные от процесса-обработчика, сравниваются с оригинальными, хранящимися в памяти управляющего процесса, и дополняют их при необходимости.

3. Система декларативных конфлюэнтных продукций

Система декларативных конфлюэнтных продукций [1] моделирует понятия предметной области (ПО) в виде индивидов и отношений. Множество правил логического вывода называется логическим модулем (ЛМ). Объект характеризуется своим именем, классом, определенностью и значением. Имя, класс и определенность задаются при описании объекта. Значение объекта определяется при вводе из файла либо в процессе выполнения модуля.

Объекты могут быть двух классов: «индивиды» и «отношения». Объекты класса «индивид» представляют индивиды ПО. Объекты класса «отношение» представляют отношения ПО.

С каждым объектом связан набор атрибутов, определяемый классом объекта. Набор атрибутов определяет область возможных значений и структуру значения объекта. Атрибутом объекта класса «индивид» является сорт объекта: «целый» либо «строковый». Областью возможных значений объекта сорта «целый» является множество целых чисел. Областью возможных значений объекта сорта «строка» являются все строки. Атрибутами объекта класса «отношение» являются число аргументов и сорта аргументов отношения: «целый» либо «строковый».

Элементы множества правил определяют взаимосвязи типа «если-то» между значениями объектов. Элементами множества правил являются правила rule вида

$$\text{rule} \equiv \text{если } Q(X) \text{ то } U_1(X_1) \,\&\, U_2(X_2) \,\&\, \ldots \,\&\, U_n(X_n),$$

где $n \in$ NAT, $Q(X)$ – формула, $U_1(X_1)$, …, $U_n(X_n)$ – простые формулы, а X, X_1, …, X_n – множества переменных.

Формула $Q(X)$, расположенная в правиле между словами если и то, называется условием правила. Формула $U_1(X_1) \,\&\, \ldots \,\&\, U_n(X_n)$, расположенная в правиле после слова то, называется следствием правила. Условие правила является логическим выражением, составленным из соотношений, положительных и отрицательных элементарных формул.

Конъюнктивным множителем следствия правила может быть соотношение, положительная и отрицательная элементарная формула.

Соотношением r является формула, составленная из двух термов, соединенных знаками отношения $<, >, =, \neq, \leq, \geq$.

Положительная элементарная формула имеет вид nr(vt), где nr – предикатный символ, vt – вектор термов. В качестве предикатного символа может выступать либо имя объекта класса «отношение». Вектор термов составляется из термов $t_1, t_2, \ldots, t_{nt}$, где $nt \in$ NAT.

Отрицательная элементарная формула имеет вид ^nr(vt), где nr – предикатный символ, vt – вектор термов.

Логическим выражением называется формула, построенная по следующим правилам:

- если l – формула, то ^ l – формула (^ – операция отрицания),
- если l1, l2 – формулы, то l1 «и» l2 – формула и l1 «или» l2 – формула,
- если l1 – формула, то и (l1) – формула.

Терм определяет способ вычисления значения. Терм может быть либо элементарным термом, либо выражением.

К элементарным термам относятся: строка, число, переменная.

Выражение составляется из элементарных термов tt_1, tt_2, …, tt_n ($n \in NAT$), соединенных знаками операций +, -, *, / и операций над строками: слияние строк, взятие подстроки, замена подстроки и др. Выражение, не содержащее переменных, будем называть выражением без переменных.

4. Распараллеливание

Так как система продукций, рассматриваемая в данной работе, является конфлюэнтной [2] (т.е. результат вычислений не зависит от порядка применения правил), и вычислительная система построена на основе многопроцессорной ЭВМ, можно использовать параллельные вычисления для обработки правил при всех существующих означиваниях.

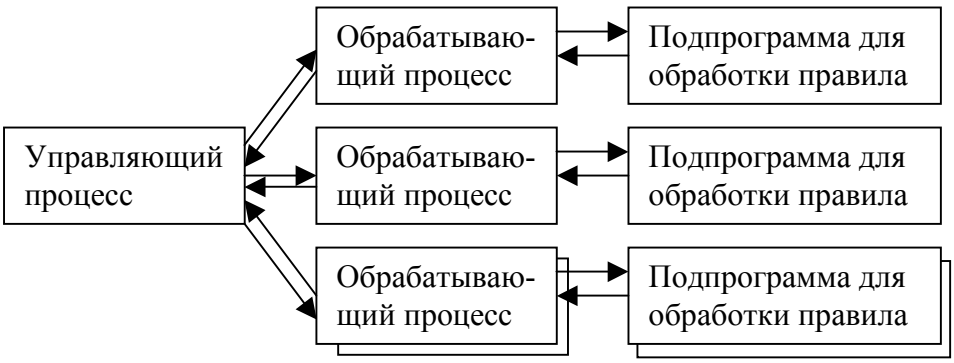

Рис. 1. Параллельная обработка правил.

В отличие от процесса логического вывода (ПЛВ), реализованного на однопроцессорном компьютере, где каждое правило обрабатывается одно за другим, многопроцессорная машина позволяет построить работу программной системы, реализующей ПЛВ, как набор процессов, выполняемых разными узлами компьютера. Одному из процессов предложена роль диспетчера, управляющего работой по подготовке правил для обработки, в то время как остальные процессы выполняют обработку каждого правила независимо друг от друга. Такое параллельное выполнение вычислений позволяет уменьшить время работы ПЛВ по сравнению с аналогичными вычислениями на однопроцессорной ЭВМ.

Управляющий процесс координирует работу с данными и правилами и организовывает запуск процессов-обработчиков. Первоначально управляющий процесс производит выборку правил базы правил и формирует множество активных правил. Затем определяется количество свободных процессов-обработчиков, которым и передаются данные, необходимые для обработки каждого правила из множества активных правил. Наконец, данные, полученные от каждого отработавшего процесса, синхронизируются с оригинальными данными. Это значит, что результат выполнения действий, описанных в следствии правила, добавляется к имеющимся данным, а само правило записывается в множество АП, если для него возможно появление новых означиваний (которые обычно появляются в том случае, когда текущее состояние ПЛВ расширяется новыми фактами).

Процесс-обработчик работает в режиме ожидания данных от управляющего процесса и ничего «не знает» о деятельности других процессов. Данные, которые он получает от управляющего процесса, несут информацию только об одном правиле и о тех объектах, которые используются в правиле. После получения данных проверяется условие применимости правила, и формируются действия, выполняемые в результате применения этого правила. Затем все выходные данные пересылаются обратно управляющему процессу.

Формальное описание распараллеленного процесса решения задачи.

Введем следующие обозначения.

МАП – множество активных правил [4] (содержит правила, для которых состояние процесса вывода содержит все необходимые для вычислений данные).

П'' – множество правил, выполненных зависимыми процессами.

q – текущее состояние процесса вывода (множество атомарных формул вида P(h), где P – термин предметной области, h – вектор скалярных констант).

$q_{-1}(\pi)$ – множество просмотренных формул в q при всех предыдущих применениях правила π ($\pi \in$ К, К – база правила), исключая данное.

$q_{-1N}(\pi)$ – новое множество просмотренных формул в q при всех предыдущих применениях правила π, включая данное.

$q_N(\pi)$ – новое множество просмотренных формул в q при данном применении правила π.

$B(P, q) = \{P(h) \mid P(h) \in q\}$ – множество формул вида P(h) из q.

$A(\{P_1, \ldots, P_l\}, q, q_1) = (B(P_1, q) \times \ldots \times B(P_l, q)) \setminus (B(P_1, q_1) \times \ldots \times B(P_l, q_1))$.

$X(\{P_1(t_1), \ldots, P_l(t_l)\})$ – множество переменных, входящих в формулы $P_1(t_1)$, …, $P_l(t_l)$.

$\lambda(x) \equiv L(x, \{P_1(t_1), \ldots, P_k(t_k)\})$ – подстановка, которая формируется следующим образом. Пусть $x = (P_1(h_1), \ldots, P_k(h_k))$ ($P_i(h_i) \in q$ для $i = 1..k$), $\{x_1, \ldots, x_m\} = X(\{P_1(t_1), \ldots, P_k(t_k)\})$. Пусть $(a_1, \ldots, a_m)$ – единственное решение системы уравнений

$$\begin{cases} t_1 = h_1 \\ \ldots \\ t_k = h_k \end{cases},$$

тогда положим $\lambda_{(x)} = \{x_1/a_1, \ldots, x_m/a_m\}$. Если система не имеет решения или оно не одно, то положим $\lambda = \varnothing$.

$$\text{Result} = \begin{cases} \{[S_1(u_1) \mid \lambda(x)], \ldots, [S_n(u_n) \mid \lambda(x)]\}, \text{если } \lambda \neq \varnothing \\ \text{и } [F(t) \mid \lambda] = \textit{истина}, \\ \ldots \\ \varnothing \text{ в противном случае} \end{cases}$$

СтартПроцесса(π, q) – операция производит запуск находящего в ожидании процесса, который будет выполнять правило π.

ПолучитьРезультат(π, М) – операция получает от отработавшего процесса результат применения правила π.

Опишем с помощью введенных обозначений процесс распараллеленного вывода. Для главного процесса P_0:

Для всех $\pi \in$ К: $q_{-1}(\pi) = \varnothing$.

Формируем начальное МАП = $\{\pi \mid A(u(\pi), q, q_{-1}(\pi)) \neq \varnothing\}$, где $u(\pi)$ – множество формул из условия правила π. П'' = $\varnothing$.

Пока ¬(МАП = ∅ и П'' = ∅) выполнить:
Пока МАП ≠ ∅ выполнить:
- выбрать $\pi \in$ МАП;
- СтартПроцесса(π, q);
- $q_{-1}(\pi) = q$;
- МАП = МАП \ {π};
конец_цикла;
Пока П''≠ ∅ выполнить:
- выбрать $\pi'' \in$ П'';
- ПолучитьРезультат(π'', M);
- $q = q \cup M$;
- если $A(u(\pi''), q, q_{-1}(\pi'')) \neq \varnothing$, то МАП = МАП $\cup$ {π' | $A(u(\pi'), q, q_{-1}(\pi')) \neq \varnothing$};
- П'' = П'' \ {π''};
конец_цикла;
конец_цикла;

Зависимый процесс P начинает работу при выполнении команды СтартПроцесса(π, q).

Пусть $w_0 = q$ – начальное состояние процесса P.

$w_1 = w_0 \cup M$; $M = \bigsqcup_{x \in A(\{P_1, \ldots, P_l\}, q, q_{-1}(\pi))} \text{Result}(x, \{P_1(t_1), \ldots, P_l(t_l)\}, \{S_1(u_1), \ldots, S_n(u_n)\}, F(t))$.

П'' = П'' $\cup \pi$.

Таким образом, на первом шаге управляющий процесс P_0 производит следующие действия:

- имея набор данных v_0, формирует МАП: $R_0 = \{r_1, r_2, \ldots, r_m\}$;
- МОЗП: $П''_0 = \varnothing$
- если $R_0 \neq \varnothing$, то для каждого правила из R_0 выбирает свободный процесс p' из $П_0$ и запускает его на наборе данных v', где v' – часть данных v_0, необходимая для применения соответствующего правила.
- затем производятся аналогичные действия по запуску свободных процессов для обработки правил из МАП до тех пор, пока не закончатся правила из МАП либо свободные процессы.

На каждом следующем шаге j процесс P_0 проверяет наличие уже отработавших процессов и, в случае их существования, получает результаты работы. Затем P_0 производит обновление данных в связи с появлением новых. Далее управляющий процесс выполняет аналогичные действия для всех оставшихся процессов из множества отработавших процессов.

Далее на шаге j процесс P_0 совершает такие же действия по формированию МАП, запуску процессов, которые были описаны выше.

Справедливо следующее утверждение: при одинаковых начальных данных конечное состояние параллельного процесса логического вывода совпадает с конечным состоянием последовательного процесса логического вывода (то есть результаты работы логического модуля в параллельном и последовательном вариантах одинаковы).

Запишем с помощью введенных обозначений правило вывода, реализованное в системе конфлюэнтных продукций РЕПРО на последовательной ЭВМ [3].

Для всех $\pi \in$ К: $q_{-1}(\pi) = \varnothing$.

Формируем начальное МАП = {π | $A(u(\pi), q, q_{-1}(\pi)) \neq \varnothing$}, где $u(\pi)$ – множество формул из условия правила π.

Пока МАП $\neq \varnothing$ выполнить:
- выбрать $\pi \in$ МАП;
- $q_N = q \cup M$, где

$$M = \bigcup_{x \in A(\{P_m, \ldots, P_m\}, q, q_{-1}(\pi))} Result(x, \{P_1(t_1), \ldots, P_m(t_m)\}, \{S_1(u_1), \ldots, S_n(u_n)\}, F(t));$$

- $q_{-1N}(\pi) = q$;
- если $A(u(\pi), q, q_{-1}(\pi)) \neq \varnothing$, то МАП = МАП $\cup$ $\{\pi' \mid A(u(\pi'), q, q_{-1}(\pi')) \neq \varnothing\}$;

конец_цикла;

5. Описание экспериментов

Для получения оценки эффективности работы схемы распараллеливания были проведены эксперименты. В каждом эксперименте измерялось время работы сгенерированных программ для последовательной ЭВМ, а также для кластера на различных наборах данных. Также исследовались время работы программ с большим количеством входных объектов (в этом случае увеличивается время работы отдельных правил), правил с малым количеством входных данных и зависимость времени работы от количества правил и количества доступных процессов. Тем самым, целью экспериментов являлось получение этих зависимостей. В экспериментах приведено среднее время для каждой ситуации.

Эксперимент №1. В данном эксперименте используется 8 правил, при этом по зависимости по данным их можно разбить на пары: 1 и 2 правила, 3 и 4, 5 и 6, 7 и 8. Между собой пары правил по данным не пересекаются, поэтому вычисления над ними можно выполнять параллельно.

При обработке примера на однопроцессорной машине вычисления заняли 1051 секунду (желтая точка на рис. 2).

При обработке примера на кластере при 9 выделенных узлах-процессорах и 9 процессах вычисления заняли 130 секунд (ситуация 1 на рис. 2).

При обработке примера на кластере при 5 выделенных узлах-процессорах и 9 процессах вычисления заняли 546 секунд (ситуаций 2 на рис. 2).

При обработке примера на кластере при 9 выделенных узлах-процессорах и 5 процессах вычисления заняли 629 секунд (ситуация 3 на рис. 2).

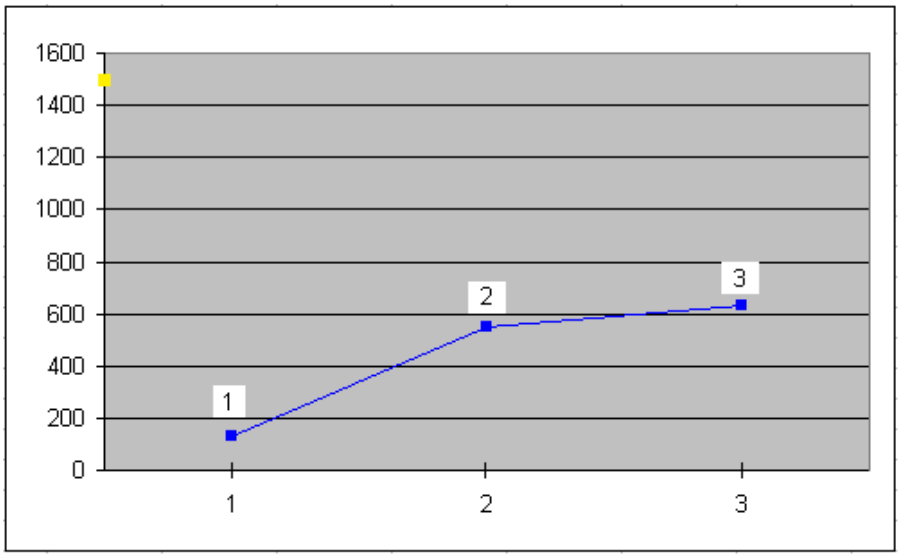

Рис. 2. Диаграмма работы программного средства при обработке 8 правил.

Из результатов видно, что параллельная обработка правил при оптимальном количестве выделенных физических узлов и таком же количестве процессов дала 8-кратный прирост скорости.

Эксперимент №2. В данном эксперименте используется 4 правила, при этом по зависимости по данным их можно разбить на пары: 1 и 2 правила, 3 и 4. Между собой пары правил по данным не пересекаются, поэтому вычисления над ними можно выполнять параллельно.

При обработке примера на однопроцессорной машине вычисления заняли 144 секунды (красная точка на рис. 3).

При обработке примера на кластере при 5 выделенных узлах-процессорах и 5 процессах вычисления заняли 96 секунд (ситуация 1 на рис. 3).

При обработке примера на кластере при 3 выделенных узлах-процессорах и 3 процессах вычисления заняли 111 секунд (ситуация 2 на рис. 3).

При обработке примера на кластере при 1 выделенном узле-процессоре и 2 процессах вычисления заняли 149 секунд (ситуация 3 на рис. 3).

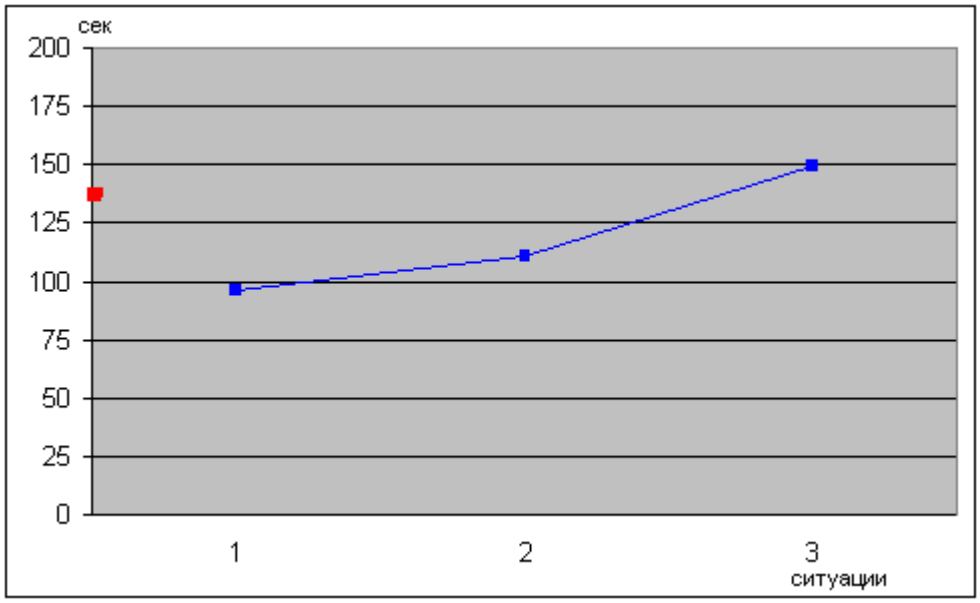

Рис. 3. Диаграмма работы программного средства при обработке 4 правил.

Из результатов видно, что параллельная обработка правил при оптимальном количестве выделенных физических узлов и таком же количестве процессов дала некоторый выигрыш в скорости.

Эксперимент №3. В данном эксперименте используется 2 правила. При вычислениях правила используют небольшое количество данных.

При обработке примера на однопроцессорной машине вычисления заняли 65 секунд (красная точка на рис. 4).

При обработке примера на кластере при 3 выделенных узлах-процессорах и 3 процессах вычисления заняли 96 секунд (ситуация 1 на рис. 4).

При обработке примера на кластере при 1 выделенном узле-процессоре и 2 процессах вычисления заняли 74 секунд (ситуация 2 на рис. 4).

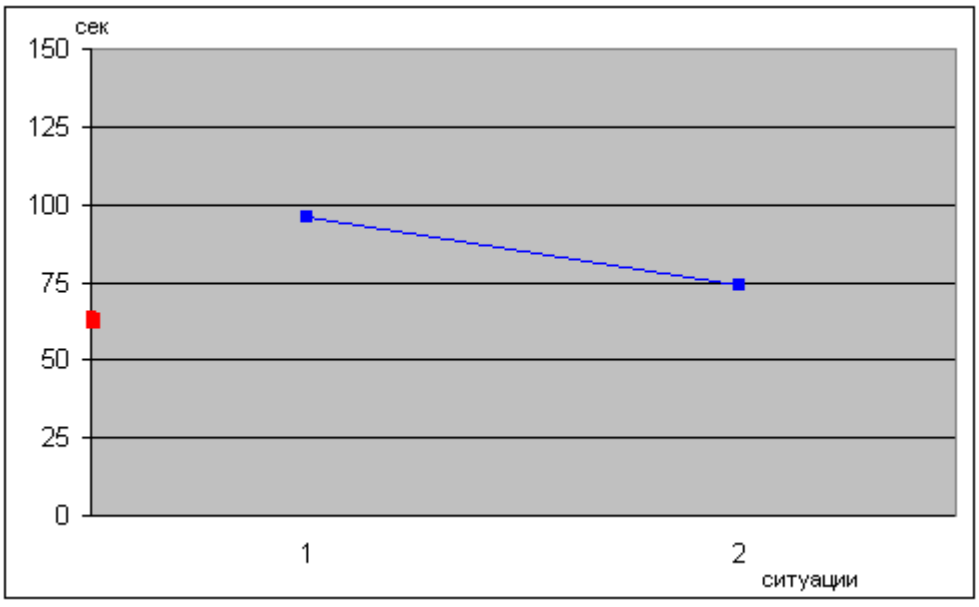

Рис. 4. Диаграмма работы программного средства при обработке 2 правил.

Из результатов видно, что параллельная обработка правил не дала выигрыша во времени из-за того, что обрабатывался незначительный объем данных, вследствие чего много времени тратилось на создание процессов и передачу данных между ними по сравнению со временем, затраченным на обработку отдельного правила.

Рассматривая результаты экспериментов, можно заметить, что наименьшее время выполнения программного средства на кластере было получено тогда, когда количество узлов и количество процессов равно количеству правил, то есть для обработки отдельного правила выделен один отдельный узел. В последнем случае меньше времени было затрачено в ситуации с одним выделенным узлом потому, что при малых объемах вычислений при работе на кластере много времени (по сравнению со временем, затрачиваемым на обработку правила) тратится на создание, передачу и прием пакета данных, необходимого для обработки правила, в соответствующий процесс.

При изучении данных, собранных во время экспериментов, можно сделать следующие выводы: при малых объемах вычислений при работе на кластере много времени (по сравнению со временем, затрачиваемым на обработку правила) тратится на создание, передачу и прием пакета данных, необходимого для обработки правила, в соответствующий процесс. За счет этого вычисления на кластере занимают намного больше времени, чем соответствующие вычисления на отдельном компьютере. Однако, по мере возрастания объема вычислений (количества правил), когда на вычисление отдельного правила тратятся десятки секунд и более, кластер начинает выигрывать во времени. При этом наименьшее время показывается тогда, когда максимальное количество процессов, выделяемых для приложения перед стартом, примерно равно количеству вызовов процессов во время исполнения приложения.

6. Заключение

В работе была рассмотрена система параллельного программирования, которая относится к классу конфлюэнтных реляционных продукций. В ней результат вычислений не зависит от порядка применения правил в процессе логического вывода. Описанная схема распараллеливания для данной системы использует тот факт, что все правила являются независимыми друг от друга, т.е. система продукций обладает естественным параллелизмом.

Схема распараллеливания, рассмотренная выше, не является единственной для системы конфлюэнтных продукций. Одной из альтернативных схем организации процесса логического вывода является схема, в которой учитываются связи между правилами по передаче данных; эти связи 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. 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 более оптимальную схему для реализации системы конфлюэнтных продукций на многопроцессорной ЭВМ.

Список литературы

1. Артемьева И.Л., Клещев А.С. Расширенная модель декларативных продукций / Препринт. ИАПУ ДВО АН СССР, 1991. 36 с.
2. Артемьева И.Л., Клещев А.С. Вывод в системах продукций с недоопределенными объектами. Процесс логического вывода / Препринт. ИАПУ ДВО РАН, 1992. 41 с.
3. Артемьева И.Л., Лифшиц А.Я., Плис Г.Я. Принципы реализации переносимого генератора экспертных систем РЕПРО // «Методы и средства создания и исследования экспертных систем». Сборник научных трудов. Владивосток: ДВО АН СССР, 1991. С. 118-129.