УДК 51:681.03

А.В. ДЕРЕВЯНЧЕНКО, А.А. ЛЯЛЕЦКИЙ

О ПРОБЛЕМЕ РАСПАРАЛЛЕЛИВАНИЯ ВЫЧИСЛЕНИЙ

Abstract: *This article is devoted to some theoretical and applied problems connected with parallel calculations. It starts from non-formal discussion up to formalizations of the notions like computing space and of a program. In particular, for formalizing the notion of program Scott's approach is used. Then, obtained formalizations are used for constructing an effective algorithm of paralleling of executing programs, which can be implemented in practice. Also, some properties of this algorithm are investigated.*
Key words: *term, algorithm, paralleling.*

Анотація: *Стаття присвячена деяким теоретичним та прикладним питанням, що стосуються розпаралелювання обчислень. Вони починаються з неформального обговорення можливих формалізацій таких понять, як обчислювальна середа та програма. У вибраної формалізації використовується поняття програми за Скоттом. Отримана формалізація використовується для конструювання ефективного алгоритму розпаралелювання виконання програм, який може бути реалізовано на практиці.*
Ключові слова: *терм, алгоритм, розпаралелювання.*

Аннотация: *Статья посвящена некоторым теоретическим и прикладным вопросам, касающихся распараллеливания вычислений. Она начинается с неформального обсуждения возможных формализаций таких понятий как вычислительная среда и программа. В выбранной формализации используется понятие программы по Скотту. Полученная формализация используется для конструирования эффективного алгоритма распараллеливания выполнения программ, который может быть реализован на практике.*
Ключевые слова: *терм, алгоритм, распараллеливание.*

1. Введение

Одной из основных проблем современного прикладного программирования является следующая задача:

Рассматриваются «сложные» вычислительное устройства U , составленные из нескольких своих вычислительных подустройств, не имеющих обязательно одинаковые мощностные характеристики. Для каждой программы P и вычислительного устройства U написать алгоритм, который во время выполнения программы P вычислительным устройством U так распределяет возникающие задания между его подустройствами, чтобы время выполнения этой программы было наименьшим.

Эта задача имеет как самостоятельный математический интерес, поскольку для её решения необходимо формализовать понятие «параллельного» вычисления, так и прикладной, поскольку задачи распараллеливания стали особо актуальными с развитием сетевых технологий и различных форм мультипроцессорности [1].

В данной работе проводится теоретическое исследование, направленное на решение поставленной задачи (или на доказательство, что в некоторых определённых смыслах такого решения не существует). Изложение материала базируется на «наивной» теории множеств. В общем случае может быть взята любая «обычная» аксиоматическая система, формализующая теорию множеств, в которой истинна аксиома выбора. Несмотря на то, что авторы стремятся решать поставленную задачу конструктивными методами, некоторые из теоретических результатов данной работы получены неконструктивным путём (например, пункт (б) теоремы 1).

2. Формализация программы в логике Скотта

Для решения поставленной задачи прежде всего надо выбрать одну из возможных формализаций понятия программы и понятия сложного вычислительного устройства (или, как мы для краткости будем говорить в дальнейшем, сложного вычислителя). Этот выбор будет проводиться в соответствии с критериями интуитивной ясности выбираемой формализации, её удобства с точки зрения описания нужного нам алгоритма и достаточной её общности.

Исходя из этого, мы будем придерживаться формализации понятия программы, предложенной в [2]. Эта формализация имеет ещё одно преимущество: в ней явно выписываются все функциональные символы; при оценке сложности с фиксированной семантикой этих символов функциям, которые им соответствуют, могут быть явно приписаны их «сложностные» коэффициенты. Кроме этого, поскольку в данной формализации явно зафиксированы синтаксис и семантика, то подобный подход можно рассматривать как логический [6].

Излагая данную формализацию, будем придерживаться следующей последовательности действий. Вначале формально изложим синтаксис (т.е. опишем алфавит и некоторый язык в нём, слова которого мы будем называть термами), далее – семантику (т.е. покажем, как при заданных значениях символов алфавита вычислять значения термов), а затем выведем взаимосвязь между подходом Скотта и уже давно ставшим классическим подходом – теорией частично рекурсивных функций.

Синтаксис определяется следующим образом. Вначале фиксируются алфавит *A*; он в нашем случае всегда состоит из множества символов, типизированных следующим образом: $f_0, f_1,...$ – функциональных символов, $p_0, p_1,...$ – предикатных символов, символов $S, C, *$, которые мы будем называть символами соответственно операции суперпозиции, операции (условного) разветвления и операции (условного) циклирования, а также из вспомогательных символов – левой скобки (правой скобки) и обычной запятой. В данном случае совокупности функциональных и предикатных символов не предполагаются не более чем счётными; тем не менее, для наглядности, при обозначении этих символов в качестве индексов будут использоваться натуральные числа.

Понятие *терма* (в алфавите *A*) определим индуктивно:

1. Каждый функциональный символ f_i есть терм.

2. Если f_i – функциональный символ и $t_1,...,t_n$ – термы, то слово $S(f_i, t_1,...t_n)$ – терм.

3. Если t_0 и t_1 – термы и p_i – предикатный символ, то $C(p_i, t_0, t_1)$ – терм.

4. Если t_0 – терм и p_i – предикатный символ, то $*(p_i, t_0)$ – терм.

5. Слово в алфавите A является термом тогда и только тогда, когда оно может быть построено только с помощью правил 1 – 4.

Грубо говоря, терм – это синтаксическая формализация понятия «метапрограммы»: фиксируя значения символов $S, C, *$ как некоторые специальные функционалы и фиксируя значения символов, входящих в запись терма, т.е. сопоставляя функциональным символам

операции, предикатным символам – предикаты и, наконец, символам отношений порядка – отношения полного порядка, мы сможем получать конкретные примеры программ. Значения всех этих символов образуют семантику рассматриваемого терма. Сейчас мы более детально покажем, какие объекты можно выбирать в качестве значений этих символов и как вычислять при этих выбранных значениях символов значение самого терма.

Зафиксируем некоторое множество M. Задать *интерпретацию функционального символа* f_i во множестве M – это значит сопоставить ему некоторую унарную, возможно, частичную операцию F_i на множестве M. Аналогично, под *интерпретацией предикатного символа* p_i во множестве M будем понимать любой унарный, возможно, частичный предикат P_i, заданный на M. Если t_0 – терм в некотором алфавите A и $\{f_{i_0},...,f_{i_m},p_{j_0},...,p_{j_n}\}$ – множество всех функциональных и предикатных символов терма t_0, а также его символов отношений порядка, то *интерпретацией терма* t во множестве M называется любое отображение ι, которое каждому функциональному символу $f_{i_j}, 1 \le j \le m$, и предикатному символу $p_{i_k}, 1 \le k \le n$, сопоставляет их некоторые интерпретации F_{i_j} и P_{i_k} в M.

Наконец, для каждого терма t_0 индукцией по его построению определим, что такое *значение терма* t_0 при его некоторой фиксированной интерпретации ι. Пусть t_0 – терм в алфавите A и ι его интерпретация в некотором множестве M. Тогда:

1) если терм t_0 имеет простейший вид, т.е. f_i, где f_i – функциональный символ, то значение терма t_0 при интерпретации ι определяется как F_i, где F_i – значение функционального символа f_i при интерпретации ι;

2) если терм t_0 имеет вид $S(t,t_1,...t_n)$, причём термы $t,t_1,...,t_n$ имеют значения при интерпретации ι, равные $T,T_1,...,T_n$ соответственно, и если, кроме того, арность операции F_i равна n, то значением терма t_0 при интерпретации ι называется функция, которая на каждом элементе m множества M принимает значение, равное $F_i(T_1(m),...,T_n(m))$, а если арность операции F_i отлична от n, то значением этого терма считается заданная над M всюду неопределённая функция;

3) если терм t_0 имеет вид $C(p_i,t_1,t_2)$, где p_i – предикатный символ со значением при интерпретации ι, равным P_i, а термы t_1 и t_2 имеют значения при интерпретации ι, равные T_1 и T_2 соответственно, то значением терма t_0 при интерпретации ι называется функция T_0, которая на каждом элементе m множества M принимает значение, равное:

а) $T_1(m)$, если элемент m удовлетворяет частичному предикату P_i;

b) $T_0(m) = T_2(m)$, если m не удовлетворяет предикату P_i;

c) неопределённости, если значение предиката P_i на m неопределено;

4) если терм t_0 имеет вид $*(p_i, t_1)$, где p_i – предикатный символ, а t_1 – терм со значениями при интерпретации ι, равными P_i и T_1 соответственно, то значением терма t_0 при интерпретации ι называется функция T_0, которая для каждого элемента m множества M определяется следующим образом: если в последовательности $m, T_1(m), T_1(T_1(m)), ..., T_1(...(T_1(m))...), ...$ элемент $T_1(...(T_1(m))...)$ - её первый такой элемент, что на нём предикат P_i истинен, то полагаем $T_0(m)$ равным значению этого элемента, если же такого элемента вида $T_1(...(T_1(m))...)$ нет, то значение $T_0(m)$ считаем неопределённым.

Назовём терм t_0 *согласованным* с его интерпретацией ι в том и только том случае, когда все подслова терма t_0, тоже являющиеся термами (т.е. так называемые подтермы), которые имеют вид $S(t, t_1, ...t_n)$, удовлетворяют такому условию: если операция F_i – интерпретация функционального символа f_i, то арность этой операции равна n.

Таким образом, как уже неформально отмечалось выше, каждый терм, рассматриваемый вместе с некоторой его фиксированной интерпретацией, выступает в качестве модели понятия программы. *Моделью* называется упорядоченная тройка $\langle M, Op, \Pr \rangle$, где M – некоторое множество, Op и $\Pr$ – совокупности заданных над ним частичных операций и частичных предикатов соответственно; множество M мы будем также называть *носителем* модели. Ясно, что если рассматривается алфавит символов *A* и если с каждым функциональным и предикатным символами этого алфавита связать соответственно частичную операцию из Op и частичный предикат из $\Pr$ модели $\langle M, Op, \Pr \rangle$, то для каждого терма над алфавитом *A* можно естественным образом однозначно определить его интерпретацию. При этом последняя называется интерпретацией терма в модели; значение терма при интерпретации, которая задаётся некоторой фиксированной моделью, мы будем называть *значением терма в модели*.

3. Взаимосвязь логики Скотта и теории частично рекурсивных функций

Укажем на взаимосвязь между предлагаемым здесь подходом и теорией частично рекурсивных функций [4].

Через θ, δ, η_m^n (m и n – натуральные числа, причём $m \le n$) будем обозначать следующие операции, арностей соответственно 1, 1 и n, заданные на натуральном ряде N:

– $\theta(k) = 0$ для любого натурального числа *k*;

– $\sigma(k) = k + 1$ для любого натурального числа *k*;

– $\eta_m^n(k_1,...,k_n) = k_m$ для любой *n*-ки натуральных чисел $\langle k_1,...,k_n \rangle$.

Операции θ, δ, η_m^n n будем называть *элементарными*. Через S, R, M обозначим операторы суперпозиции, примитивной рекурсии и минимизации соответственно. В соответствии с классическим определением операция g, заданная на натуральном ряде N, называется *частично рекурсивной* в классе операций F над N тогда и только тогда, когда g может быть получена из элементарных операций и из операций из F с помощью операторов суперпозиции, примитивной рекурсии и минимизации; если при этом $F = \varnothing$, то функция g также называется просто частично рекурсивной. Отметим, что значения символа S в формализации Скотта (когда рассматриваются модели с натуральным рядом в качестве носителя) и в теории частично рекурсивных функций совпадают.

Если рассматривается модель, носитель которой состоит из всевозможных *n*-ок натуральных чисел при нефиксированном n и совокупность операций которой содержит все элементарные операции, то ясно, что каждой частично рекурсивной функции можно сопоставить терм и модель так, что последние будут задавать эту частично рекурсивную функцию: при этом оператору суперпозиции отвечает он сам, оператору минимизации отвечает оператор циклирования и, наконец, оператор примитивной рекурсии, очевидно, выражается через суперпозицию, разветвление и циклирование. Наоборот, если зафиксирована такая модель, причём *Op* обозначает её совокупность всех операций и задан некоторый терм, то функция, которая задаётся этим термом интерпретацией в рассматриваемой модели, можно сопоставить частично рекурсивную относительно *Op* функцию.

4. Построение формальной модели параллельных вычислительных сред

И, наконец, всё готово к формализации и решению поставленной в самом начале задачи. Прежде всего, потребуется формализовать понятие сложного вычислителя и понятие сложности программы. Это будет происходить следующим образом: фиксируется некоторое положительное натуральное число n, которое интерпретируется как количество всех «простейших» вычислителей, составляющих первоначальное вычислительное устройство U. Также будет рассмотрен случай, когда n есть «плюс бесконечность», т.е. первое бесконечное ординальное число ω. Далее, если заданы алфавит *A*, причём F и P обозначают множества всех функциональных и предикатных символов, входящих в *A*, и модель *M*, в которой интерпретируются все «интерпретируемые» символы из этого алфавита, то под *сложностной функцией* будем понимать любую функцию $s : \{1,...,n\} \times (F \cup P) \to N$ (если $n = \omega$, то эта функция s имеет вид $N \times (F \cup P) \to N$). Эту функцию можно понимать как такую, которая сопоставляет каждому функциональному символу f_i (предикатному символу p_i) и «простейшему» вычислителю с номером n время, которое займёт вычисление каждого значения частичной операции F_i (частичного предиката P_i), где F_i – это интерпретация символа f_i (P_i – это интерпретация символа p_i) в рассматриваемой модели.

Упорядоченную четвёрку $\langle A, M, n, s \rangle$, где *A* – алфавит; M – модель, интерпретирующая символы алфавита *A*; n – элемент множества $(N \setminus \{0\}) \cup \{\omega\}$, и, наконец, s – это сложностная функция, будем называть *вычислительной средой*.

Вычислительная среда $\langle A, M, n, s \rangle$ называется *монотонной* тогда и только тогда, когда сложностная функция s монотонна по второму аргументу, т.е. для любой пары натуральных чисел m_0, m_1, не больших чем n, и для любых функциональных символов f и g алфавита *A* из $s(m_0, f) \le s(m_0, g)$ следует $s(m_1, f) \le s(m_1, g)$ и для любых предикатных символов *p* и *q* алфавита *A* из $s(m_0, p) \le s(m_0, q)$ следует $s(m_1, p) \le s(m_1, q)$.

Опишем теперь процесс вычисления терма на основном, «сложном» вычислителе. Для этого обычно вводят так называемые состояния [3], затем вводят понятие конфигурации, а сам процесс вычисления терма определяется как последовательность конфигураций, удовлетворяющая некоторым специальным условиям. Но прибегнем к другой схеме. Откажемся формально вводить понятие состояния, правда, в некотором смысле состояниями можно считать моменты времени – такты работы «сложного» вычислителя, однако, с нашей точки зрения, одним из основных свойств состояний является способность использовать их либо при записи терма, либо при определении понятия его вычисления. Вместо этого, каждой вычислительной среде, каждому терму в ней и каждому входному значению сопоставим специальное помеченное дерево, которое при этих входных значениях указывает, когда и сколько раз должны вычисляться интерпретации вхождений функциональных и предикатных символов в рассматриваемый терм. Затем определим понятие конфигурации и, наконец, основываясь на этом понятии и на упоминаемых выше помеченных деревьях, определим, что такое вычисления терма.

Под *помеченным деревом* будем понимать упорядоченную пару вида $\langle \Delta, \theta \rangle$, где Δ – «обычное» дерево (с фиксированным корнем), а θ – любая функция, определённая на вершинах дерева Δ. Элементы множества значений θ будут называться *метками* помеченного дерева $\langle \Delta, \theta \rangle$. Если Δ – дерево, то для любого множества a под $\Delta \times \{a\}$ будем понимать (единственное) дерево, изоморфное Δ относительно взаимно однозначного отображения, которое каждой вершине v дерева Δ относит пару $\langle v, a \rangle$.

Пусть $S = \langle A, M, n, s \rangle$ – вычислительная среда, t_0 – терм в алфавите *A* и $\pi = m_0, m_1, \ldots$ – счётная последовательность элементов носителя модели M. Индукцией по длине терма t_0 построим специальное помеченное дерево $\langle \Delta, \theta \rangle$, метками которого являются пары-элементы множества $(F \cup P \cup Terms(S)) \times N$, где $Terms(S)$ обозначает множество всех термов в вычислительной среде S (при этом будем считать, что строящееся дерево «растёт снизу вверх»):

1) если терм t_0 имеет простейший вид, т.е. f_i, где f_i – функциональный символ, а F_i обозначает интерпретацию символа f_i в модели M, то полагаем Δ состоящей из единственной вершины, которой функция θ сопоставляет пару $\langle F_i,1\rangle$;

2) если терм t_0 имеет вид $S(f_i,t_1,...t_n)$, причём символ операции f_i интерпретируется как F_i и для термов $T_1,...,T_n$ уже построены помеченные деревья $\langle\Delta_1,\theta_1\rangle,...,\langle\Delta_n,\theta_n\rangle$, то объединяем деревья $\Delta_1\times\{1\},...\Delta_n\times\{n\}$ под общей вершиной $\{0\}$, и для каждой вершины v_i дерева Δ_i полагаем $\theta(\langle v_i,i\rangle)=\theta_i(v_i)$ и $\theta(a)=\langle F_i,1\rangle$;

3) если терм t_0 имеет вид $C(p_i,t_1,t_2)$, где p_i – предикатный символ со значением при интерпретации ι, равным P_i, а термы t_1 и t_2 имеют значения при интерпретации ι, равные T_1 и T_2, которым соответствуют помеченные деревья $\langle\Delta_1,\theta_1\rangle$ и $\langle\Delta_2,\theta_2\rangle$, то полагаем:

а) $\Delta=\Delta_1\times\{1\}\cup\{0\}$, причём вершина 0 расположена «ниже» всех остальных вершин дерева Δ, и $\theta(0)=\langle P_i,1\rangle$, $\theta(\langle v_1,1\rangle)=\theta_1(v_1)$ для любой вершины v_1 из Δ_1, если значение выражения $P_i(m_0,m_1,...,m_k)$ суть истинна (*k* – арность предиката P_i);

б) $\Delta=\Delta_2\times\{1\}\cup\{0\}$, причём вершина 0 расположена «ниже» всех остальных вершин дерева Δ, и $\theta(0)=\langle P_i,1\rangle$, $\theta(\langle v_2,1\rangle)=\theta_2(v_2)$ для любой вершины v_2 из Δ_2, если значением выражения $P_i(m_0,m_1,...,m_k)$является ложь;

в) $\langle\Delta,\theta\rangle=\langle\varnothing,\varnothing\rangle$, если значение выражения $P_i(m_0,m_1,...,m_k)$ не определено;

4) если терм t_0 имеет вид $*(p_i,t_1)$, где p_i – предикатный символ, t_1 – терм со значениями при интерпретации ι, равными P_i и T_1 соответственно, причём T_2, рассматриваемый как функция, имеет арность n и ему соответствует помеченное дерево $\langle\Delta_2,\theta_2\rangle$, то полагаем:

а) $\Delta=\Delta_2\times\{2\}\cup\{0\}\cup\{1\}$ причём вершина 1 расположена «ниже» всех остальных вершин дерева Δ, кроме 0, $\theta(\langle v_2,2\rangle)=\theta_2(v_2)$ для любой вершины v_2 из Δ_2, $\theta(0)=\langle p_i,k\rangle$ и $\theta(1)=\langle T_i,k\rangle$, где k – такой наименьший номер элемента последовательности

$$P_i(m_0,m_1,...,m_{n-1}),P_i(T_1(m_0,m_1,...,m_{n-1}),m_1,...,m_{n-1}),$$
$$P_i(T_1(T_1(m_0,m_1,...,m_{n-1}),m_1,...,m_{n-1}),m_1,...,m_{n-1}),...,$$

которому соответствует элемент, равный истине;

б) $\langle\Delta,\theta\rangle=\langle\varnothing,\varnothing\rangle$, если такого натурального числа k не существует.

Построенное помеченное дерево $\langle\Delta,\theta\rangle$ мы будем называть *деревом вычисления терма* t_0 в вычислительной среде S при интерпретации, определяемой последовательностью π.

Пусть F и P обозначают множества всех функциональных и предикатных символов алфавита *A*, а *N* обозначает множество всех натуральных чисел.

Конфигурационной функцией вычислительной среды $\langle A, M, n, s\rangle$ будет называться любая частичная функция k, которая действует из множества $\{1,...,n\}$ во множество $(F \cup P \cup \{\#\}) \times N$, и такая, что для любого $m, 1 \le m \le n$, $k(m)$ имеет значение $\langle x, 0\rangle$ тогда и только тогда, когда $x = \#$; понятие *конфигурации* будем отождествлять с понятием конфигурационной функции. Каждая конфигурационная функция k показывает, какой «простейший» вычислитель, какую операцию или предикат, сопоставленный функциональным символам алфавита *A*, выполняет в некоторый момент времени; если F_i – операция, соответствующая функциональному символу f_i, и $k(m_i) = \langle f_i, t_i\rangle$, то t_i означает время, которое необходимо потратить m_i-му «простейшему» вычислителю, чтобы завершить выполнение операции F_i (аналогично для предикатов). При этом, равенство $t_i = 0$ означает, что в данный некоторый момент времени ни один из «простейших» вычислителей не выполняет никакую из операций F_i и предикатов P_i.

Пусть S – вычислительная среда, t – терм, $\langle \Delta, \theta\rangle$ – произвольное дерево вычисления терма t_0 в S и $k = k_0, k_1, ...$ – произвольная, не более чем счётная последовательность конфигураций. Для каждого вхождения s некоторого символа из множества $(F \cup P)$ в терм t пусть $\langle \Delta_s, \theta_s\rangle$ обозначает дерево выполнения наименьшего подтерма терма t, которое содержит вхождение s. (В дальнейшем будем отождествлять вхождение символа в терм и сам этот символ.) Положим $D_k(0) = \varnothing$, $D_k(1)$ равным множеству всех меток листов дерева $\langle \Delta, \theta\rangle$, и для каждого её индекса последовательности k, строго большего 1, через $D_k(i)$ обозначим множество

$$\left\{q \in F \cup P \,\middle|\, \exists j \exists m\left(j < i \;\&\; k_j(m) = \langle q', 1\rangle \;\&\; q \textit{ непосредственно больше } q' \text{ в дереве } \langle \Delta_q, \theta_q\rangle\right)\right\}$$

Содержательно множества D_k можно описать следующим образом. Если каждый элемент рассматриваемой последовательности k конфигураций интерпретировать как последовательность «состояний» простейших вычислителей в процессе выполнения некоторого терма, то каждое D_k, кроме 0-го, является множеством всех операций, функций и термов либо уже выполненных, либо доступных для выполнения.

Пусть t – терм в алфавите *A*; тогда *вычислением терма* t_0 в вычислительной среде $S = \langle A, M, n, s\rangle$ на входных данных, определяемой последовательностью $\pi = m_0, m_1, ...$ называется любая не более чем счётная последовательность конфигурационных функций $k = k_0, k_1, ...$, которая удовлетворяет следующим условиям:

1) функция k_0 совпадает с функцией тождественно элементу $\langle \#, 0\rangle$;

2) для каждого положительного натурального числа i последовательность $k_0, k_1, ..., k_i$ удовлетворяет условиям 1 и 2, причём если k_i не является последним элементом последовательности π, то k_{i+1} суть любая такая функция, что для каждого натурального положительного числа m, не большего чем n:

а) из $k_i(m) = \langle q, s \rangle, q \in F \cup P$ и $s > 1$ следует $k_{i+1}(m) = \langle q, s-1 \rangle$;

б)из $k_{i+1}(m) = \langle q, s(m,q) \rangle$ следует

$$k_i(m) = \langle q', s' \rangle, s' \in \{0,1\}, q' \in F \cup P \cup \#, q \in D_k(i+1) \setminus \left(\cup \{ D_k(j) \mid j < i \} \right);$$

3) i является последним индексом последовательности P тогда и только тогда, когда $\cup \{ D_k(i) \mid 1 \le m \le i \} = F$ и из $j < i$ следует $\cup \{ D_k(j) \mid 1 \le m \le j \} = F$ (если же такого числа i не существует, то последовательность является счётной).

Сложностью вычисления терма называется длина (т.е. число всех индексов) соответствующей последовательности k. Нас интересует алгоритм, который для каждой вычислительной среды по входному терму и его входным данным (т.е. последовательности π) строит его вычисление наименьшей длины.

5. Алгоритм распараллеливания выполнения программ и его свойства

Пусть $\langle A, M, n, s \rangle$ – вычислительная среда, t_0 – терм в алфавите *A* и π – входные данные для этого терма и $T = \langle \Delta, \theta \rangle$ соответствующее дерево вычислений терма t_0. *Весом* $W(V)$ каждой ветви V такого дерева T будем называть число, определяемое по формуле $\sum \{ W(v) \cdot k(v) \mid v \in V \}$, где $k(v)$ – такое число, что $\langle v, k(v) \rangle \in T$, причём если $v \in F \cup P$ v ∈ F ∪ P, то $W(v) = s(v,1)$.

Пусть $S = \langle A, M, n, s \rangle$ – монотонная вычислительная среда, вычислители которой расположены по убыванию эффективности их работы. Если известно, что для каждого из подтермов t_i вида $*(p_i, t_i)$ терма t_0 соответствующий цикл длится не более чем I_i итераций, то рассматриваем следующий процесс, который будет называться ***алгоритмом распараллеливания вычисления терма*** (или просто алгоритмом распараллеливания) и обозначать через **A**:

1) строим дерево $\langle \Delta', \theta' \rangle$, получающееся из дерева $\langle \Delta, \theta \rangle$ выполнения t_0 в S путём замены всех его меток вида $\langle p_i, k \rangle$ и $\langle f_j, k \rangle$ (т.е. вершин, «отвечающих» циклам) на вершины $\langle p_i, I_i \rangle$ и $\langle f_j, I_i \rangle$ соответственно;

2) в дереве $\langle \Delta', \theta' \rangle$ выбираем все его максимальные ветви (т.е. ветви, которые не содержаться ни в каких других ветвях, отличных от самих себя) и располагаем их в список S_1 в соответствии с их весом относительно этого дерева;

3) полагаем $k_0(m) = (\#, 0)$ для каждого $m, m \le n$;

4) если последовательность конфигурационных функций $k_0, k_1, ..., k_i$ уже построена, T' – поддерево дерева $\langle \Delta', \theta' \rangle$, и «есть» «свободные» вычислители, т.е. существуют такие m, что $k_1(m) = 1$, то вычислитель с таким минимальным числом m сопоставляем, «загружаем» самой нижней первой ветви в списке S_{i+1} операцией или предикатом, затем отнимаем от приписанного к нему справа числа единицу и заново упорядочиваем список (преобразованных) максимальных ветвей дерева в этом дереве или предикатом; повторяем так до тех пор, пока не останется «свободных» вершин. Таким образом мы однозначно определили k_{i+1};

5) если последовательность $k_0, k_1, ..., k_i$ является вычислением терма t_0, то алгоритм заканчивает свою работу, иначе переходит к пункту 4.

Для каждого алгоритма **B** через t_B обозначим функцию, которая сопоставляет каждому «входному данному» нашего алгоритма «время», которое занимает выполнение нашего алгоритма на этом входном данном.

Теорема 1. В каждой монотонной вычислительной среде S алгоритм распараллеливания является самым эффективным в следующих смыслах:

(а) если оценки $I_1, ..., I_n$ являются точными, то сложность вычисления каждого терма является наименьшей в смысле нашего определения (т.е. вычисление термов происходит «быстрее всего»);

(б) для любого другого алгоритма **B**, удовлетворяющего пункту (а), существует такая нумерация входных данных и такая линейная функция h, что $t_A(d_{i+1}) \div t_B(d_{i+1}) \le h(t_A(d_i) \div t_B(d_i))$, где d_i – входное данное с номером i, а $\div$ – это так называемая усечённая разность: бинарная операция на множестве натуральных чисел, которая каждой паре $\langle m, n \rangle$ натуральных чисел сопоставляет их разность $m - n$, если $m \ge n$, и равна нулю, в противном случае

Доказательство пункта (а) ведётся индукцией по построению терма; доказательство пункта (б) имеет неконструктивный характер и существенным образом опирается на теорему компактности для разбиений [5], и, следовательно, на аксиому выбора.

Также можно показать, что если вычислительная среда S не является монотонной, то предложенный алгоритм является неверным. Выполнение $k = k_0, k_1, ...$ терма t в вычислительной среде $S = \langle A, M, n, s \rangle$ называется *предоптимальным* в том и только том случае, когда:

1) мощность множества $\left\{m\middle|k_1(m)\neq\langle\#,0\rangle\right\}$ совпадает с числом, равным минимальному из чисел n и количеству всех листочков дерева выполнения терма t в S;

2) для каждого натурального числа $i(1<i<n)$ выполняется равенство $\left|\left\{m\middle|k_1(m)\neq\langle\#,0\rangle\right\}\right|=\min\left\{n\middle|D_k(i)\setminus\left(\cup\left\{D_k(j)\middle|j<i\right\}\right)\right\}$.

То есть, предоптимальное выполнение терма – это такое выполнение терма, что в каждый момент времени количество всех загруженных простых вычислителей является максимальным.

Используя изложенную формализацию, применённую к первоначальной задаче, также не трудно убедиться в следующем утверждении:

Теорема 2. При условии, что имеются точные оценки $I_1,...,I_n$ количеств итераций, поставленная задача в общем случае (когда вычислительная система S является произвольной) является, с одной стороны, разрешимой, а, с другой, для любого алгоритма, решающего её, существует такая вычислительная среда S, что сумма сложностей его выполнения и (любого) предоптимального выполнения каждого терма строго меньше, чем минимальная (т.е. «самая» эффективная) сложность выполнения этого терма.

Интуитивно это можно объяснить так: чтобы самым эффективным образом «распараллелить» выполнение терма, нужно прежде его (произвольным образом) выполнить. Чтобы доказать этот факт, достаточно в качестве контр- примера для утверждения, обратного второй части теоремы 2, рассмотреть предложенный алгоритм.

В случае, когда значения оценок $I_1,...,I_n$ неизвестны или они сильно отличаются от точных значений, и даже предполагая рассматриваемую вычислительную среду S монотонной, нельзя построить «по существу» самый эффективный алгоритм.

Теорема 3. Не существует алгоритма, который по (монотонной) вычислительной системе по терму «распараллеливает» (не имея оценок $I_1,...,I_n$) выполнение этого терма в системе и который удовлетворяет пункту (б) теоремы 1.

Для доказательства следует рассмотреть наш алгоритм, обратив внимание на то, что в нём оценки $I_1,...,I_n$ дать невозможно, не зная аналогичных оценок для входного терма этого алгоритма.

6. Выводы

Полученные результаты позволяют моделировать параллельные системы, которые понимаются в достаточно широком смысле. В частности, они могут включать в себя различные мультипроцессорные системы [7], а также сетевые топологии. Предложен эффективный алгоритм распараллеливания термов (программ) для монотонных (т.е. однородных) сред, который может быть реализован на практике. Изучены теоретические свойства этого алгоритма, а также всего класса алгоритмов, которые решают задачу, поставленную во введении. В частности, оказалось, что для того, чтобы эффективно распараллелить некоторый терм, необходимо знать достаточно

много информации о нём и о системе, в которой будет проводиться его параллельное вычисление. Теорема 1 утверждает, что в этом случае возможно самое эффективное распараллеливание. Если же информации недостаточно, то теорема 3 указывает, что тогда универсального алгоритма распараллеливания не существует. Всё это указывает на то, что при практическом конструировании той или иной программы программист должен выбрать из следующей альтернативы. С одной стороны, построение универсальной распараллеливающей системы, равно как и программы, которая позволяет эффективно распараллеливать вычисления, часто является сама по себе довольно трудной задачей, поскольку в этом случае необходимы точные оценки сложности работы этой программы. С другой стороны, если программа, которую планируется моделировать [8], будет использоваться в каждой вычислительной системе малое число раз и сложность её выполнения является относительно небольшой, время, затраченное на эффективное распараллеливание, плюс её (параллельное) выполнение может оказаться меньше, чем просто её выполнение. В связи с этим, с практической точки зрения, на первый план выходит задача построения оценок сложности работы распараллеливающих алгоритмов. Это авторы планируют сделать в следующих работах.

СПИСОК ЛИТЕРАТУРЫ

1. Шоу А. Логическое проектирование операционных систем. – М.: Мир, 1981. – С. 69 –136.
2. Scott D.S. The Lattice of Flow Diagrams // Lecture Notes in Mathematics. – 1971. – N 188. – С. 311–366.
3. Глушков В.М., Цейтлин Г.Е., Ющенко Е.Л. Алгебра. Языки. Программирование. – К.: Наукова думка, 1989. – 376 с.
4. Мальцев А.И. Алгоритмы и рекурсивные функции. – М.: Наука, 1986. – 368 с.
5. Protasov I.V. Combinatorics of numbers. – Lviv: VNTL, 1997. – 70 с.
6. Редько В.Н., Гришко В.Н., Редько И.В. Дескриптологическая среда программирования // Проблемы программирования. – 2002. – № 1–2. – С. 7 – 14.
7. Воеводин В. Параллельные вычисления. – М.: BHV, 2002. – 603 с.
8. Анисимов А.В., Деревянченко А.В. Построение виртуального параллельного пространства с использованием технологии ПАРУС-JAVA // Тезисы конференции ІБС, ШІ. – Дивноморское. – 2003. – 22–27. – С. 18 – 19.