СТРУКТУРНЫЙ СИНТЕЗ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

В.Э. Малышкин

Институт вычислительной математики
и математической геофизики СО РАН,
Новосибирск

Метод структурного синтеза программ имеет в настоящее время все шансы для хорошей реализации на нынешних мультикомпьютерах. В первую очередь потому, что способ представления знаний в нём – вычислительные модели, семантические сети – хорошо соответствует структуре мультикомпьютеров.

В зависимости от характера формализации ПО можно выделить несколько подходов к проблеме синтеза программ. Наиболее цельный из них в теоретическом отношении – *логический синтез* программ. При этом подходе формальным описанием ПО является некоторое достаточно полное математическое исчисление, представляющее закономерности предметной области. Задача синтеза ставится как задача нахождения доказательства теоремы существования, а программа извлекается из этого доказательства. К сожалению, этот подход весьма сложно реализуем на практике, так как, с одной стороны, для нетривиальных предметных областей создание полной непротиворечивой теории не осуществимо. С другой стороны, еще недостаточно развиты методы машинного доказательства теорем, особенно если ищется программа, обладающая нужными свойствами, например оптимальная по какому-либо критерию.

Эти трудности заставляют пользоваться при описании ПО менее богатыми, но более удобными для машинного восприятия средствами. Одно из них заключается в наведении на ПО некоторой структуры, отражающей ассоциативные связи между понятиями ПО. Явное выражение этих связей позволяет свести случайный поиск к ассоциативному, что решающим образом сказывается на оценках трудоемкости процесса планирования. Кроме того, структурное описание имеет наглядное графическое представление, удобное для восприятия и обработки. Описанный подход к синтезу программ получил название *структурного синтеза* программ.

Проблема синтеза программ на вычислительных моделях рассматривается в статической постановке: по представлению вычислительной модели, выделенной группе переменных, служащих для ввода начальных данных, и другой группе переменных, служащих для вывода результатов, требуется найти программу, вычисляющую по заданным входным данным

соответствующие результаты (если они для этих входных данных определены).

При одной и той же постановке задачи можно синтезировать различные программы.

Принципиально важно, что синтезируются программы наилучшего качества. Основными критериями качества являются надежность, оптимальность (в ряде смыслов), степень параллелизма и т.п. В целом в процессе синтеза программы решается несколько задач, укладывающихся в следующую общую схему.

Пусть даны:

– класс S спецификаций входных заданий;

– класс P результирующих выходных программ;

– отношение эквивалентности $\sim$ на P;

– отношение качества $>$ на P, удовлетворяющее аксиомам частичного порядка.

Требуется найти алгоритм $A{:}S\to P$ такой, что результат синтеза – программа $p=A(s)$, $p\in P$ – удовлетворяет спецификации $s\in S$ и является наилучшей в смысле $>$ в классе программ $\{p|p\sim A(s)\}$.

Таким образом, проблема синтеза заключается в разработке алгоритма, конструирующего по каждому элементу из S некоторый элемент из P, наилучший среди всех программ, решающих специфицированную задачу. Метод синтеза параллельных программ на вычислительных моделях отличает высокая прагматичность. Прагматичность метода основана на том, что вычислительная модель является, собственно, собранием хороших решений различных задач предметной области и решение новой задачи собирается (компонуется) из заготовленных хороших решений. В таких условиях есть надежда сконструировать на их основе хорошее решение новой задачи. Конечно, иногда для этого приходится включать в вычислительные модели новые необходимые решения.

Базой знаний в вычислительных моделях является множество алгоритмов и фрагментов программ, причем хороших алгоритмов. И комбинации хороших алгоритмов тоже могут быть хороши. Они хотя и не обязательно оптимальны, но и не самые худшие. Задача вывода приемлемого алгоритма становится простой и сводится к ограниченному управляемому перебору на графе. Однако качество сконструированной программы, разнообразие программ существенно зависят от размеров фрагментов. С уменьшением размеров фрагментов растет разнообразие и теоретически достижимое качество программ (увеличивается множество заготовленных алгоритмов) и одновременно быстро растёт сложность конструирования программы (растет число комбинаций алгоритмов), так что на текущий

момент приходится использовать весьма большие фрагменты и небольшой перебор.

В дополнение к этому, так же, как массив джунглей разбивается тропинками на фрагменты, так и описание предметной области разбивается в вычислительных моделях на множество меньших предметных областей, для которых построение более или менее полных теорий (для каждой подобласти своей) более вероятно. Таким образом, в вычислительных моделях формализованное описание предметной области строится как система теорий, связанных соотношениями модели.

Основные определения

Пусть заданы:

1) конечное множество $\boldsymbol{X}=\{x, y, ..., z\}$ переменных для представления вычисляемых и измеряемых величин;

2) конечное множество $\boldsymbol{F}=\{a, b, ..., c\}$ функциональных символов (операций) арности $m\times n$, $m\geq 0$, $n\geq 0$, m и n, вообще говоря, различные для разных операционных символов;

3) с каждым символом операции a арности $m\times n$ связан набор $\mathrm{in}(a)=(x_1, ..., x_m)$ входных и набор $\mathrm{out}(a)=(y_1, \ldots y_n)$ выходных переменных, при этом $i\neq j \rightarrow y_l \neq y_j$.

Пара $\boldsymbol{C}=(\boldsymbol{X},\boldsymbol{F})$ называется *простой вычислительной моделью* (ПВМ). Операция $a\in\boldsymbol{F}$ описывает возможность вычисления переменных $\mathrm{out}(a)$ из переменных $\mathrm{in}(a)$ с помощью некоторой процедуры. Модели удобно изображать графически (рис. 1).

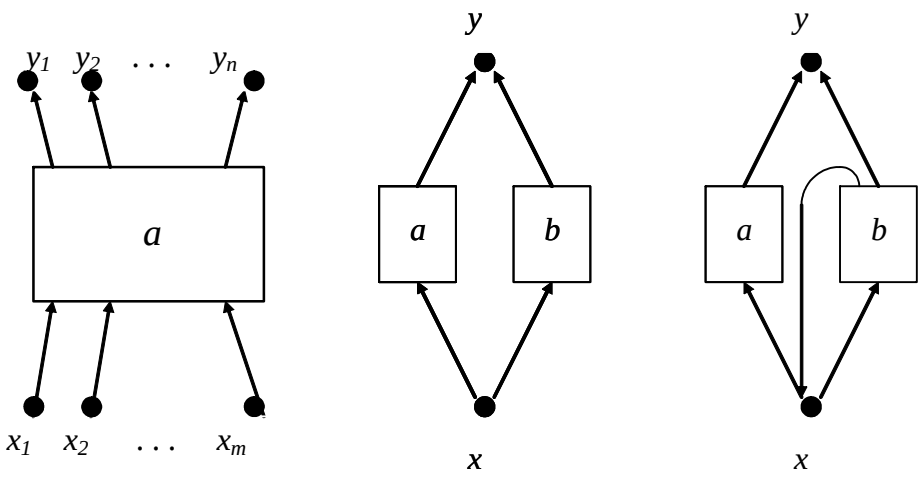


Рис. 1. Определение множества термов $T(V,F)$, порожденных V и F ($V\subseteq\boldsymbol{X}$, $F\subseteq\boldsymbol{F}$)

Основным объектом теории является функциональный терм. Формальное определение терма и его атрибутов задается следующими правилами.

1) Если $x \in V$, то x есть терм t, $x \in T(V,F)$.

2) Пусть t^1, ..., t^s входят в $T(V,F)$. Пусть $a \in F$, $\text{in}(a)=(x_1,...,x_s)$. Тогда терм t, равный $a(t^1,...,t^s)$, включается во множество $T(V,F)$ тогда и только тогда, когда выполняется $\forall i\ (x_i \in \text{out}(t^i))$.

Интерпретация

Пусть $V \subseteq \boldsymbol{X}$ – некоторое подмножество переменных, играющее роль входных переменных алгоритма. *Интерпретация* $\boldsymbol{I}$ в области интерпретации D есть функция, которая сопоставляет:

– каждой переменной $x \in V$ элемент $d_x=\boldsymbol{I}(x) \in D$, d_x называется значением переменной x в интерпретации $\boldsymbol{I}$ (далее просто значение);

– каждой операции $a \in \boldsymbol{F}$ вычислимую функцию f_a: $D^m \to D^n$, $\text{in}(a)=\{x_1, x_2, ..., x_m\}$, $\text{out}(a)=\{y_1, y_2, ..., y_n\}$;

– каждому терму $t=a(t_1,t_2,...,t_m)$ суперпозицию функций в соответствии с правилом $\boldsymbol{I}(a(t_1,t_2,...,t_m))=f_a(\boldsymbol{I}(t_1),\boldsymbol{I}(t_2),...,\boldsymbol{I}(t_m))$.

Пусть $t=a(t_1,t_2,...,t_m)$ – произвольный терм такой, что $\text{in}(a)=\{x1, x2, ..., xm\}$, $\text{out}(a)=\{y1, y2, ..., yn\}$. Тогда набору переменных $\text{out}(a)$ сопоставляется в интерпретации $\boldsymbol{I}$ набор (кортеж) значений

$$val(t)=(d_1,d_2,...,d_n)=f_a(val_{x1}(t_1),val_{x2}(t_2),...,val_{xn}(t_n)).$$

Здесь $val_z(t)$ обозначает значение того компонента набора $val(t)$, который соответствует переменной z.

Впредь всегда предполагается, что каждой функции $f_a=\boldsymbol{I}(a)$ соответствует модуль mod_a, который и будет использоваться в программе для вычисления f_a. Если существуют два различных алгоритма вычисления переменной y (существуют два различных терма t_1 и t_2 таких, что $y \in \text{out}(t_1) \cap \text{out}(t_2)$, $\text{in}(t_1) \cup \text{in}(t_2) \subseteq V$) из одних и тех же начальных переменных V, то по смыслу модели эти два значения переменной y должны быть эквивалентны (равны в простейшем случае). Например, площадь треугольника может быть вычислена различными алгоритмами (различными термами), однако вычисленные значения должны быть равны (если, конечно, начальные данные действительно определяют треугольник и запрограммированные модули верно вычисляют функции). Это одна из причин, по которой следует строить избыточные вычислительные модели, в которых существует несколько алгоритмов вычисления переменных.

Дальше будем работать только с такими *корректными* интерпретациями. По определению, в корректной интерпретации для любой переменной y и любых термов t_1 и t_2 таких, что $y \in \text{out}(t_1) \cap \text{out}(t_2)$, значения $val_y(t_1)=val_y(t_2)$, т.е. оба терма *вырабатывают* одно и то же значение переменной y. Это позволяет говорить о значениях переменных, вычисляемых не только отдельными термами, но и множествами термов.

Все термы из множества $T(V,F)$ обладают свойством: $t\in T(V,F)\rightarrow \text{in}(t)\subseteq V$. Термы множества $T(V,F)$ определяют все вычисления (все алгоритмы), которые могут быть произведены из переменных множества V (если заданы их значения).

Определение. Терм называется *избыточным*, если на пути от входных переменных к вершине терма (на ветви расширенного дерева, представляющего терм) одна и та же операция встречается более одного раза для вычисления одной и той же своей выходной переменной.

Множество всех не избыточных термов из $T(V,F)$ обозначается $T1$, $T1\subseteq T(V,F)$. Впредь будем работать только с термами из $T1$.

Определение. Множество термов $T_V^W=\{t\in T1 \mid \text{out}(t)\cap W\neq\varnothing\}$. Это множество задает все вычисления, которые основаны на V и завершаются в W.

Определение. Множество термов $R\subseteq T_V^W$ такое, что $\forall x\in W$ $\exists t\in R(x\in\text{out}(t))$ называется (V,W)-планом вычислений.

Такой (V,W)-план вычислений определяет алгоритм решения задачи: на C вычислить W из V.

Планирование алгоритма

Пусть задана ПВМ $\boldsymbol{C}=(\boldsymbol{X},\boldsymbol{F})$, которая после трансляции представлена в виде двух таблиц *ТХ* и *ОП*. Каждая строка таблицы *ТХ* имеет вид $(x,A(x),\text{comp}(x))$, а таблицы *ОП* – $(a,\text{in}(a),\text{out}(a))$, $x\in\boldsymbol{X}$, $a\in\boldsymbol{F}$, $\text{comp}(x)=\{a\in\boldsymbol{F} \mid x\in\text{out}(a)\}$, $A(x)=\{a\in\boldsymbol{F} \mid x\in\text{in}(a)\}$.

Чтобы не делать далее несущественных оговорок, предполагается, что $\text{in}(a)\neq\varnothing$ для любых $a\in\boldsymbol{F}$. Алгоритм планирования состоит из двух частей: восходящей и нисходящей.

В *восходящей* части алгоритма строятся множества переменных и операций, используемых в термах из множества $T_V=T(V,F)$.

Обозначим $V_0=V$, тогда

$$F_0=\{a\in\boldsymbol{F} \mid \text{in}(a)\subseteq V_0\}=\bigcup_{x\in V_0}\{a\in A(x) \mid \text{in}(a)\subseteq V_0\}$$

содержит все операции ПВМ такие, что $\text{in}(a)\subseteq V_0$. Далее формируется множество $V_1=\{x\in\boldsymbol{X} \mid x\in\text{out}(a)\wedge a\in F_0\}\cup V_0$, на основе V_1 строится множество

$$F_1=\bigcup_{x\in V_1\setminus V_0}\{a\in A(x) \mid \text{in}(a)\subseteq V_1\}$$

и т. д. до тех пор, пока при некотором целом положительном k не окажется, что $F_k=\varnothing$. На этом завершается восходящая часть алгоритма планирования. Множества V_i и F_I, $i=0,...,k$ содержат все переменные и операции, используемые в термах из множества T_V.

Если $W\not\subseteq V_k$, то планирование можно прекращать, так как в этом случае существует переменная в W, которая не вычисляется никаким термом

из множества T_V, и, следовательно, не существует алгоритма решения сформулированной задачи на основе имеющихся знаний о ПО. В этом случае говорим, что сформулированная задача синтеза *не разрешима*. В противном случае можно начать строить множества переменных и операций, используемых в термах из T_V^W. Обозначим $F^*=\bigcup_{i=0} F_i$ и определим множества

$$G_i = \bigcup_{x \in H_{i-1}} \left\{ a \in F^* \middle| a \in \mathrm{comp}(x) \wedge a \notin \bigcup_{m=1}^{i-1} G_m \right\}, \quad H_i = \bigcup_{a \in G_i} \mathrm{in}(a).$$

Построение множеств G_I и H_i завершается, когда при некотором целом положительном r окажется $G_r = \varnothing$. Множества G_I и H_i, $i = 1, ..., r$ содержат все переменные и операции, используемые в термах из множества T_V^W. Кроме того, в них остаются операции и переменные, необходимые для построения некоторых дублирующих вычислений.

Таким образом, результатом планирования является ПВМ, оставшаяся от ***C*** после удаления из *ТХ* и *ОП* «лишних» переменных и операций. Множество T_V^W не строится, подходящий в некотором смысле (V,W)-план T строится в каждом конкретном случае процедурой выбора алгоритма.

В случае, когда $W \not\subseteq V_k$, сформулированная задача синтеза оказывается неразрешимой и необходимо изменить формулировку задачи, т. е. либо уменьшить W, удалив из него невычислимые переменные, либо расширить V, включив в него такие новые переменные, что станут вычислимыми все переменные из W. Для уменьшения затрат на расширение V может быть использован алгоритм планирования. Для этого необходимо выполнить его *нисходящую* часть из множества переменных $W'=W\backslash V_k$ с использованием всех операций из F. Все переменные из построенных при этом множеств H_i, i=1, 2, ..., r являются кандидатами на включение в V. Из них человек может выбрать те переменные, значения которых ему доступны. Нетрудно также построить человеко-машинные алгоритмы, помогающие сделать такой выбор.

Как можно было бы определить вычисления на вычислительной модели? Как определено интерпретацией. Вначале вносятся значения во все переменные из V. Затем выполняются все операции из F_1 (выполняются модули mod_a для всех $a \in F_1$), при этом (в соответствии с определением интерпретации) вычисляются значения переменных из $V_1\backslash V_0$. Далее выполняются все операции из F_2 и т.д. до F_k. Ясно, что такие вычисления могут быть весьма неэффективными, поскольку будут реализованы все алгоритмы вычисления W, которые определены моделью, – и самые хорошие и самые плохие. Кроме того, могут быть вычислены и переменные, которые не входят в W и которые не надо вычислять вообще.

Основной смысл планирования именно в том и состоит, чтобы до начала вычислений статическим анализом

а) удалить операции и переменные, не участвующие в вычислении *W*,

и

б) подготовить информацию для конструирования приемлемого по качеству или даже оптимального алгоритма вычисления *W*.

Это, конечно, существенная и высокоэффективная оптимизирующая процедура. Известно, что наибольший оптимизационный эффект при построении решения задачи извлекается именно на этапе конструирования алгоритма. На всех последующих этапах эффект оптимизации значительно ниже.

Литература

1. Вальковский В.А., Малышкин В.Э. Синтез параллельных программ и систем на вычислительных моделях. – Новосибирск: Наука, 1988. – 128 с.

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ В ЗАДАЧАХ ИССЛЕДОВАНИЯ И ПРОГНОЗА ПОГОДЫ*

А.В. Старченко

Томский государственный университет

Одним из современных способов исследования и прогноза развивающихся над ограниченной территорией локальных атмосферных процессов является математическое моделирование, опирающееся на использование мезомасштабных метеорологических моделей [1–3]. Эти модели, как правило, включают нестационарные трехмерные уравнения гидродинамики и тепломассообмена и отличаются различными подходами параметризации атмосферных процессов: потоков коротковолновой и длинноволновой радиации, образования конвективных систем, формирования пограничного слоя, микрофизики влаги, развития турбулентности атмосферы, тепло- и влагообмена в подстилающей поверхности [1].

Теоретическую основу для использования методов математического моделирования для исследования и прогноза погоды заложил в 1904 г. норвежский ученый В. Бьеркнес, который определил задачу прогноза погоды как задачу решения уравнений атмосферной термогидродинамики. Впервые численные методы для исследования и прогноза погоды были применены в 1922 г. Л. Ричардсоном. В 1939 г. К.-Г. Россби продемонстрировал применение линеаризованных уравнений гидродинамики к предсказанию погоды. Используя в своей работе допущение, что скорость воздуха в атмосфере не зависит от вертикальной координаты, ему удалось показать, что эволюция поля скорости может быть описана решением уравнения для завихренности. Однако первый успешный численный прогноз погоды был получен Дж. Чарни с соавторами лишь в 1950 г. на первом электронном компьютере ENIAC на основе модели Россби, в которую была добавлена нелинейная адвекция. Существенным шагом в теории прогноза явились работы Г. И. Марчука, Н. И. Булеева и К. Хинкельмана, в которых впервые учитывалось влияние процессов на большой площади на изменение атмосферных условий в пункте, для которого рассчитывается прогноз.

В современной классификации математические модели для исследования и прогноза атмосферных явлений делятся по пространственно-временным масштабам на глобальные, мезо- и микромасштабные. Глобальные модели ориентированы на изучение процессов с характерными горизонтальными масштабами от нескольких до сотен тысяч километров и

* Работа выполнена при поддержке РФФИ, гранты 04-07-90219, 05-05-98010р_обь.

временными масштабами от нескольких дней до нескольких недель. К таким процессам относят глобальную циркуляцию, синоптические циклоны и антициклоны, атмосферные фронты и пр. Моделируемая область, как правило, покрывает либо всю поверхность земного шара, либо его полусферу. Мезомасштабными или локальными называют модели, описывающие процессы с пространственным масштабом от единиц до нескольких тысяч километров и характерным временем от часа до нескольких дней. Это крупномасштабная турбулентность, динамика кучевых облаков, грозы, бризовые и горно-долинные циркуляции, городской «остров тепла». К микромасштабным относят модели, предназначенные для исследования атмосферных процессов с характерными размерами, не превышающими нескольких километров: движение воздуха в уличных каньонах и вблизи отдельно стоящих зданий. Как правило, такие модели призваны решать задачи определения воздействия источников загрязнений (промышленные объекты, транспорт) в элементах городской застройки с целью снижения техногенной нагрузки.

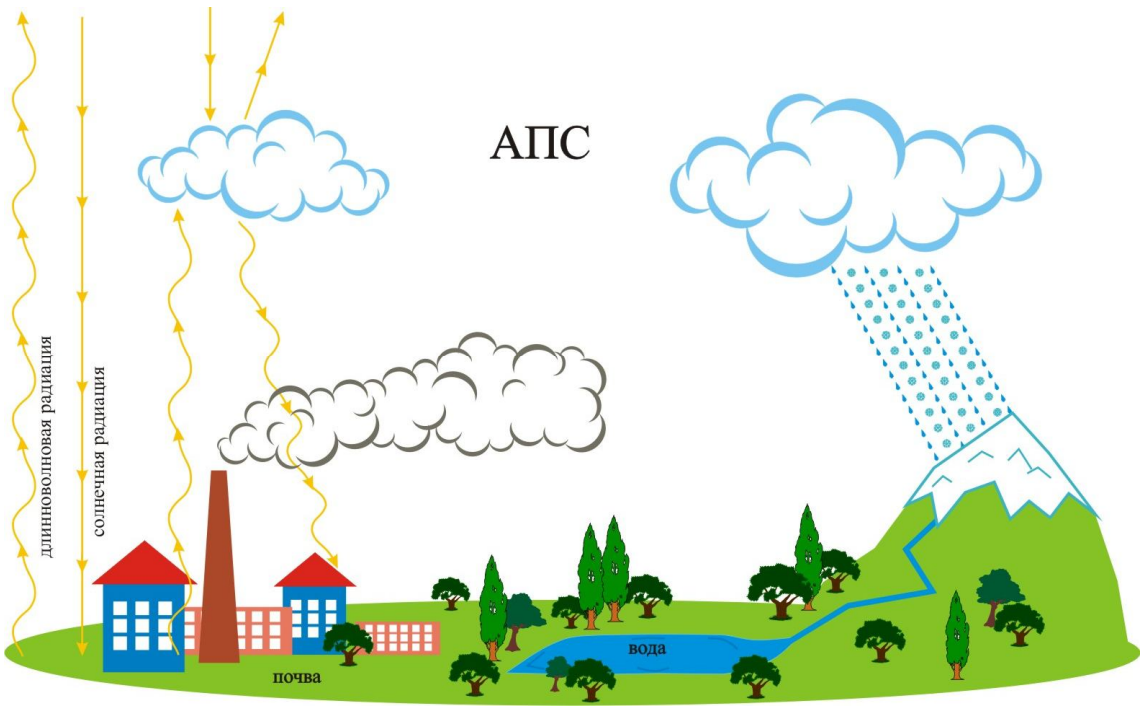


Рис. 1. Процессы и факторы, учитываемые в современных мезомасштабных моделях

Появившиеся в 60-х годах XX в. мезомасштабные модели атмосферы активно развивались последние четыре десятилетия и в настоящее время успешно применяются для решения различных прикладных задач: прогноз локальной погоды, изучение формирования атмосферных циркуляций, образования туманов и облачности, распространения примесей и их трансформации. Сегодня мезомасштабная метеорологическая модель – матема-

тическая модель для расчета погоды над ограниченной территорией с достаточно высоким разрешением. Мезомасштабные модели применяются там, где условия окружающей среды изменяются существенно во времени и пространстве. Мезомасштабные модели включают, как правило, нестационарные трехмерные уравнения гидротермодинамики и отличаются различными подходами параметризации атмосферных процессов и учета взаимодействия с подстилающей поверхностью (рис. 1).

Основные уравнения мезомасштабных моделей

Уравнение неразрывности

$$\frac{\partial \rho}{\partial t}+\frac{\partial(\rho u)}{\partial x}+\frac{\partial(\rho v)}{\partial y}+\frac{\partial(\rho w)}{\partial z}=0. \qquad (1)$$

Уравнения движения:

$$\rho\left(\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}+w\frac{\partial u}{\partial z}\right)=-\frac{\partial p}{\partial x}+\rho f v+K_H\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)+\frac{\partial}{\partial z}\left(K_Z^m\frac{\partial u}{\partial z}\right); \qquad (2)$$

$$\rho\left(\frac{\partial v}{\partial t}+u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}+w\frac{\partial v}{\partial z}\right)=-\frac{\partial p}{\partial y}-\rho f u+K_H\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right)+\frac{\partial}{\partial z}\left(K_Z^m\frac{\partial v}{\partial z}\right); \qquad (3)$$

$$\rho\left(\frac{\partial w}{\partial t}+u\frac{\partial w}{\partial x}+v\frac{\partial w}{\partial y}+w\frac{\partial w}{\partial z}\right)=-\frac{\partial p}{\partial z}-\rho g+K_H\left(\frac{\partial^2 w}{\partial x^2}+\frac{\partial^2 w}{\partial y^2}\right)+\frac{\partial}{\partial z}\left(K_Z^m\frac{\partial w}{\partial z}\right). \qquad (4)$$

Уравнение баланса энергии

$$\rho\left(\frac{\partial \theta}{\partial t}+u\frac{\partial \theta}{\partial x}+v\frac{\partial \theta}{\partial y}+w\frac{\partial \theta}{\partial z}\right)=K_H\left(\frac{\partial^2 \theta}{\partial x^2}+\frac{\partial^2 \theta}{\partial y^2}\right)+\frac{\partial}{\partial z}\left(K_Z^h\frac{\partial \theta}{\partial z}\right)+R+\Phi_\theta. \qquad (5)$$

Уравнение влагообмена

$$\rho\left(\frac{\partial q}{\partial t}+u\frac{\partial q}{\partial x}+v\frac{\partial q}{\partial y}+w\frac{\partial q}{\partial z}\right)=\frac{\partial}{\partial x}\left(K_Z^h\frac{\partial q}{\partial x}\right)+\frac{\partial}{\partial y}\left(K_Z^h\frac{\partial q}{\partial y}\right)+\frac{\partial}{\partial z}\left(K_Z^h\frac{\partial q}{\partial z}\right)+\Phi_q. \qquad (6)$$

Уравнение состояния

$$p=\rho RT,\ R=R_0\left[\frac{1-q}{M_{air}}+\frac{q}{M_{H_2O}}\right];\ \ \theta=T(p_0/p)^{R/c_p}. \qquad (7)$$

Вычислительные технологии

В настоящее время решение уравнений (1)–(7) может быть получено лишь численно с использованием высокопроизводительных компьютеров. Вычислительные технологии отыскания приближенного решения опираются, как правило, на конечно-разностные методы, позволяющие перейти от дифференциальной задачи в области непрерывного изменения аргументов к конечно-разностной постановке (системе линейных уравнений) на сеточной области. Обычно в мезомасштабных моделях применяется сетка, равномерная по широте и долготе и не равномерная по высоте. Вычислительные уровни модели сгущаются при приближении к поверхности на основе подходящего преобразования [1,2]. После выбора сетки производится замена (аппроксимация) дифференциальных операторов (1)–(7) конечно-

разностными соотношениями, при этом используются схемы аппроксимации высоких порядков – второго или третьего по времени и второго – шестого по координатам. При численной реализации задачи (1)–(7) обычно применяются явно-неявные разностные схемы с расщеплением для учета быстропротекающих процессов, таких как звуковые волны, выпадение осадков и т.п. Некоторые слагаемые уравнений (1)–(7) аппроксимируются неявными разностными схемами для ослабления требований вычислительной устойчивости, выражающихся в ограничении временного шага численного интегрирования системы нелинейных уравнений (1)–(7). К таким слагаемым относятся члены, представляющие вертикальные звуковые волны и вертикальную диффузию. Получающиеся системы сеточных уравнений имеют трехдиагональную матрицу и решаются экономичным методом прогонки [4].

При численной реализации мезомасштабных моделей также применяются неявные разностные схемы, которые разрешаются на основе метода покоординатного расщепления или расщепления по физическим процессам [2] или высокоскоростных итерационных методов [3]. Использование неявных разностных схем при моделировании атмосферных процессов с высоким разрешением является более предпочтительным, поскольку снимается существенное ограничение на шаг интегрирования уравнений (1)–(7) по времени. В то же время применение неявных разностных схем требует больших вычислительных затрат вследствие необходимости обращения матрицы системы линейных уравнений, которые могут быть компенсированы путем использования высокопроизводительной вычислительной техники.

Параллельная реализация алгоритмов численного решения задачи (1)–(7) опирается на одномерную или двумерную декомпозицию сеточной области, которая проводится по горизонтальным координатным направлениям. На каждом шаге по времени параллельные вычисления осуществляются одновременно и независимо в каждой выделенной сеточной подобласти, однако для обеспечения расчетов на следующих временных слоях требуется пересылка данных соседним сеточным подобластям, связанным используемым шаблоном сетки. На многопроцессорной вычислительной технике с распределенной памятью время передачи одного числа между вычислительными узлами может на порядки превосходить время одной арифметической операции. Поэтому увеличение доли вычислений в общем объеме при одновременном сокращении временных затрат параллельной программы является одной из важнейших задач при проведении расчетов на мультикомпьютерах с локальной памятью [5].

Используемые мезомасштабные модели

В данной работе представлены результаты применения мезомасштабных метеорологических моделей MM5 [6], WRF [7] и модели ТГУ-ИОА

СО РАН [3] к исследованию развития региональных атмосферных процессов над территорией юга Западной Сибири с использованием кластерных систем ТГУ и ИОА СО РАН.

Разработанная в Национальном центре атмосферных исследований США в кооперации с университетом штата Пенсильвания метеорологическая моделирующая система пятого поколения MM5 (Mesoscale Model 5) [6] предназначена для исследования и прогноза локальных и региональных атмосферных процессов на мезо-бета- и мезо-гамма-масштабах (2–220 км). Негидростатическая версия модели, в которой используется следящая за поверхностью координатная система, позволяет варьировать горизонтальное разрешение от сотен метров до десятков километров. Для достижения высокого разрешения в MM5 включена возможность организации проведения расчетов во вложенных областях с односторонним или двухсторонним влиянием (рис. 2). Разработаны версии модели для многопроцессорной вычислительной техники. Модель тестировалась на суперкомпьютерах с различной архитектурой (Compaq Cluster, CrayT3E, HP SPP, IBM SP, SGI Origin, SUN, PC Linux Cluster) на основе двумерной декомпозиции сеточной области. Версия MM5 для МВС с распределенной памятью базируется на использовании стандарта MPI.

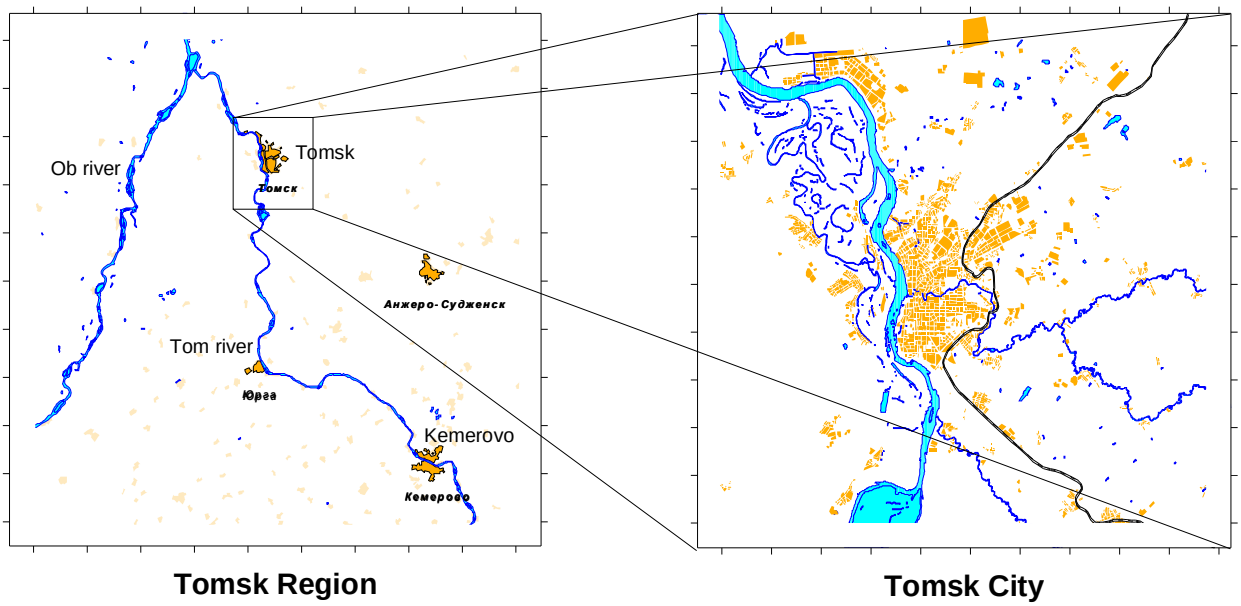


Рис. 2. Организация расчетов во вложенных областях с целью повышения разрешающей способности мезомасштабной модели

В результате расчетов MM5 генерирует метеорологические поля (горизонтальные и вертикальную компоненты вектора скорости ветра, давление, температуру, влажность воздуха, характеристики облачности и осадков, потоки тепла, влаги и количества движения, потоки коротковолновой и длинноволновой радиации и т.п.). Модель MM5 имеет большое количе-

ство схем параметризации подсеточных физических процессов, выбираемых пользователем, исходя из требований поставленной задачи и пространственного масштаба моделируемых процессов.

В настоящее время ММ5 установлена на веб-портале «АТМОС» (http://climate.atmos.iao.ru) и вычислительном кластере ИОА СО РАН, находится в открытом доступе и используется для коллективного решения задач физики атмосферного пограничного слоя для ограниченной территории (рис. 3).

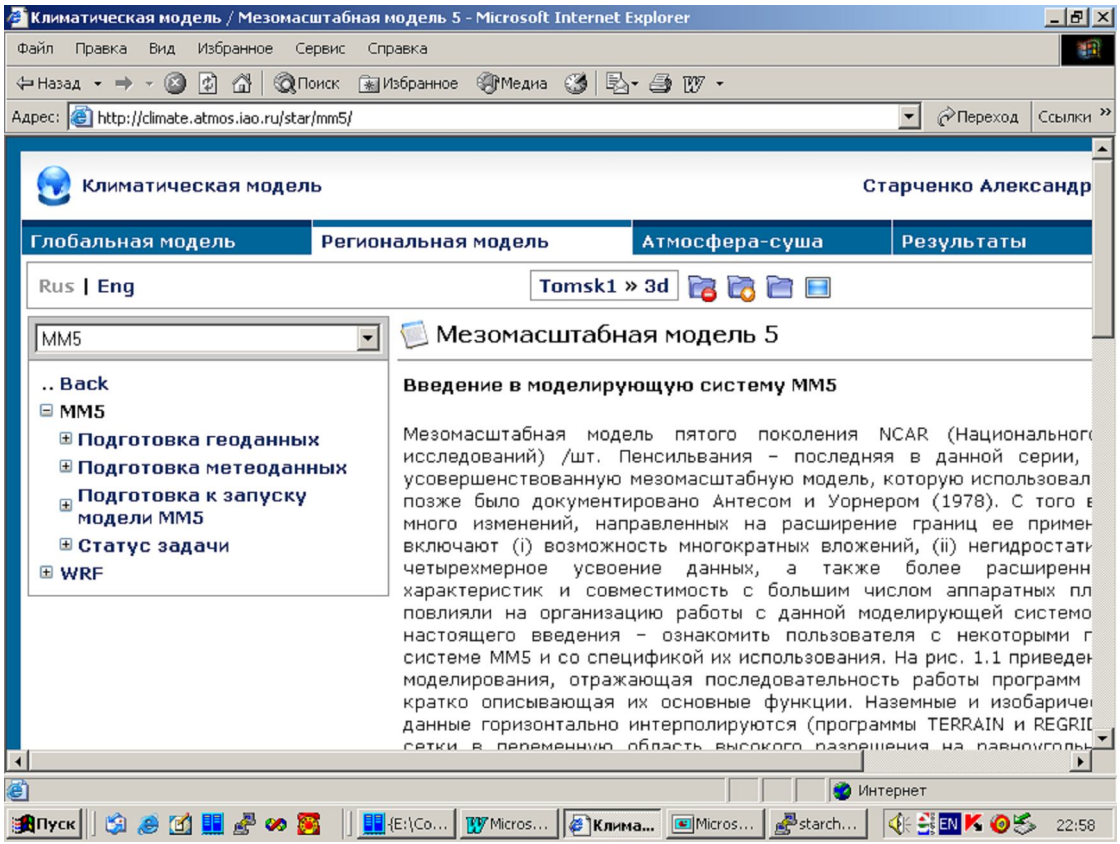


Рис. 3. Пользовательский интерфейс для запуска мезомасштабной модели ММ5 на вычислительном кластере ИОА СО РАН

Модель для исследования и прогноза погоды WRF (Weather Research and Forecast) разработана в Национальном центре атмосферных исследований США [7]. WRF-модель включает негидростатические уравнения для сжимаемого газа, учитывает неоднородность поверхности. В модели реализована технология организации расчетов во вложенных областях с односторонним или двухсторонним влиянием. Используются зависимая от поверхности система координат, сгущающиеся к поверхности Земли сетки.

WRF может использоваться при решении широкого круга задач в масштабах от сотен метров до тысяч километров, включая идеализированные течения, а также при численном предсказании погоды в реальном времени. Кроме того, WRF-модель может использоваться как динамическое

ядро в компьютерных моделирующих системах, предназначенных для исследования переноса примеси и анализа качества атмосферного воздуха над урбанизированными территориями. В модели WRF предлагается большое количество схем параметризации, которые можно совмещать в любом виде. Есть параллельная версия модели для многопроцессорных систем с общей или распределенной памятью. Параллельные вычисления на основе WRF-модели проводились на IBM SP системах, HP/Compaq Alpha рабочих станциях, PC Linux кластерах, Sun Solaris и SGI Origin суперкомпьютерах.

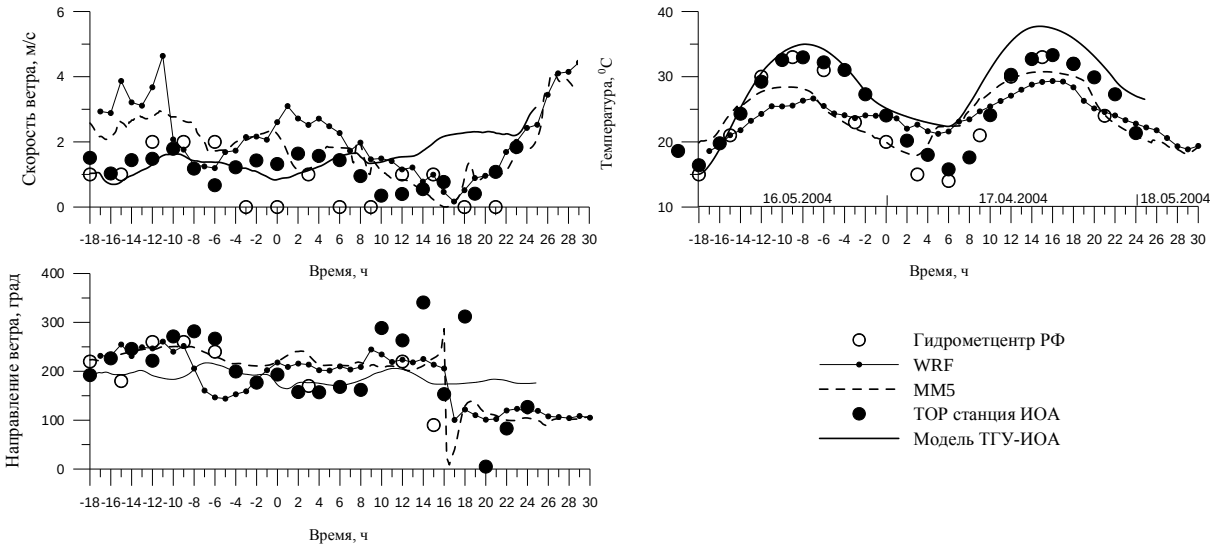


Рис. 4. Изменение скорости и температуры приземного воздуха в Томске

В Томском государственном университете и Институте оптики атмосферы СО РАН создана компьютерная моделирующая система для анализа распределения метеорологических параметров и концентрации примесей в атмосферном пограничном слое и их осаждения на поверхности с неоднородными свойствами [3]. Эта система применяется для параллельных расчетов на МВС с распределенной памятью важных с точки зрения оценки качества воздуха метеорологических параметров (направление и скорость ветра, турбулентная структура атмосферного воздуха) над урбанизированными территориями относительно небольших горизонтальных масштабов со сложным рельефом. Для повышения точности прогноза используется технология организации расчетов во вложенных областях.

Для географической привязки этих моделей (выбор области исследования, учет рельефа и распределения категорий землепользования подстилающей поверхности) используются топографические данные с высоким разрешением. Задание первого приближения метеорологических полей реализуется с помощью архивных данных либо данных предшествующего прогноза. В качестве архивных данных берутся данные реанализа Нацио-

нального центра охраны окружающей среды США или Европейского центра среднесрочных прогнозов погоды.

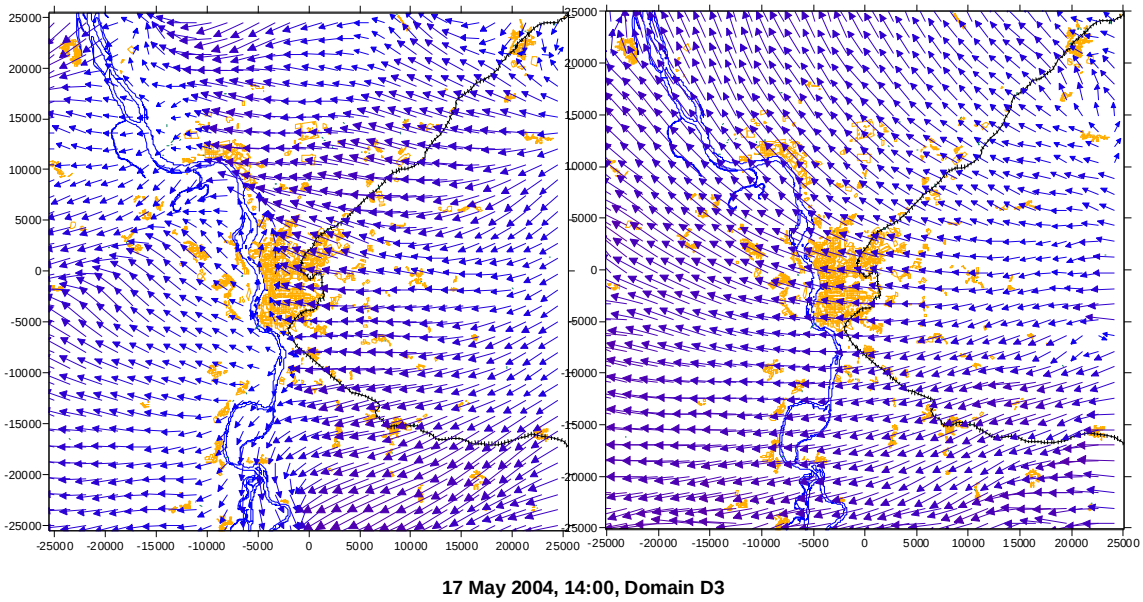


Рис. 5. Векторное поле скорости приземного ветра над Томским районом на 20 часов 17.05.2004 г. Слева – MM5, справа – WRF

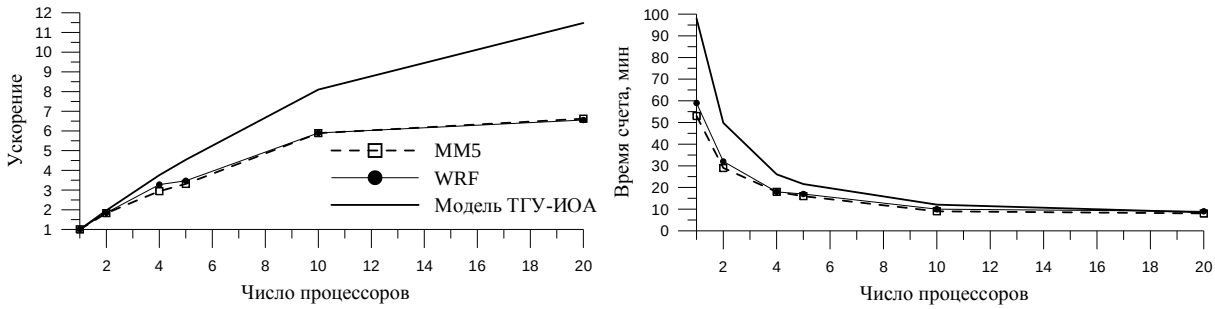


Рис. 6. Зависимость ускорения вычислений и времени счета от числа используемых процессоров

Рассматриваемые модели, опирающиеся на данные измерений и расчеты по глобальным моделям, прошли всестороннее тестирование для различных погодных условий для юга Западной Сибири и Томского района. На рис. 4 для периода времени 16–17 мая 2004 г. представлены результаты сравнения предсказанных значений скорости и направления приземного ветра, а также температуры воздуха на высоте 2 м в г. Томске (85^0 в.д., $56{,}5^0$ с.ш.) с данными наблюдений, выполненных Гидрометцентром РФ (http://meteo.infospace.ru) и на ТОР станции ИОА СО РАН (http://meteo.iao.ru). Использовались вычислительные сетки 52x52x31. Из рисунка видно, что выбранные для анализа метеорологические модели показывают хорошее качество воспроизведения направления и скорости приземного ветра. На рис. 5 приведено векторное поле ветра в приземном слое

воздуха для 20 часов 17 мая 2004 г., предсказанное с использованием моделей MM5 и WRF.

На рис. 6 приведены графики изменения ускорения и времени счета одного часа периода моделирования при различном числе используемых процессоров вычислительного кластера ИОА СО РАН. Ускорение рассчитывалось как отношение времени счета, полученного на одном процессоре, к времени счета многопроцессорного варианта программы. Как видно из рисунка, при использовании до десяти процессоров наблюдается практически линейный рост ускорения и значительное (в несколько раз) сокращение времени счета. Далее при увеличении числа применяемых процессоров эффективность параллельной программы снижается, что можно объяснить, в первую очередь, особенностью архитектуры многопроцессорной системы с распределенной памятью, вычислительные узлы которой представляют двухпроцессорные функциональные устройства, а также тем, что при фиксированном общем объеме вычислительной работы при увеличении числа применяемых процессоров также увеличивается и доля временных затрат на пересылку данных. Тем не менее применение кластера позволяет существенно сократить временные затраты на получение решения.

Литература

1. Pielke R.A. Mesoscale Meteorological Modeling. – Orlando: Academic Press, 1984. – 622 p.
2. Пененко В.В., Алоян А.Е. Модели и методы для задач охраны окружающей среды. – Новосибирск: Наука, 1985. – 224 с.
3. Старченко А.В. Численное моделирование городской и региональной атмосферы и оценка ее влияния на перенос примеси // Вычислительные технологии. – 2004. – Т. 9, ч. 2. – С. 98−107.
4. Самарский А.А., Николаев Е.С. Методы решения сеточных уравнений. – М.: Наука, 1978. – 591 с.
5. Старченко А.В., Есаулов А.О. Параллельные вычисления на многопроцессорных вычислительных системах – Томск: Изд-во Том. ун-та, 2002. – 56 с.
6. Dudhia J. A nonhydrostatic version of the Penn State/NCAR mesoscale model: validation tests and simulation of an Atlantic cyclone and cold front // Monthly Weather Review. – 1993. – Vol. 121. – P. 1493–1513.
7. Klemp J.B., Wilhelmson R. The simulation of three-dimensional convective storm dynamics // Journal of Atmospheric Sciences. – 1978. – Vol. 35. – P. 1070–1096.

РАСЧЕТНЫЕ НЕСТРУКТУРИРОВАННЫЕ СЕТКИ ДЛЯ РАСПРЕДЕЛЕННЫХ ВЫЧИСЛЕНИЙ

С.П. Копысов, А.Б. Пономарев, В.Н. Рычков, С.Н. Зубцовский

Институт прикладной механики УрО РАН, г. Ижевск

Введение

Эффективность распараллеливания вычислительных алгоритмов определяется программной реализацией сетки в тех случаях, когда алгоритмы включают шаги, связанные с изменением топологии и расчетных данных сетки в процессе вычислений.

В работе [1] была предложена объектно-ориентированная модель расчетной неструктурированной сетки. Целью данной работы является создание на ее основе распределенной модели расчетной сетки, предназначенной для реализации параллельных алгоритмов. Одним из требований является внесение минимально возможного числа изменений в существующую модель сетки. Модель распределенной расчетной сетки является основой для решения большинства расчетных задач на многопроцессорных системах.

1. Использование расчетных сеток на многопроцессорных системах

Расчетная сетка – это данные, над которыми выполняются операции двух видов: методы модификации сетки и вычислительные алгоритмы, основанные на сеточной аппроксимации. Первые используют сетку как входные и выходные данные и модифицируют ее. Вторые же – абстрагируются от сетки к системе уравнений; их основные действия выполняются над матрицами и векторами, которые формируются в соответствии с сеточной аппроксимацией. В вычислительных алгоритмах с адаптивным перестроением сетки [2, 3] эти виды операций комбинируются.

Если рассматривать расчетные сетки с точки зрения их использования на многопроцессорных системах, то это распределенные данные. Над ними выполняются последовательные и параллельные операции. В общем степень распараллеливания вычислительных алгоритмов не зависит от программной реализации сетки. Но как только в алгоритмах появляются шаги, связанные с модификацией сетки (перестроение, повторное использование после изменения числа расчетных параметров в отдельных элементах, балансировка нагрузки и пр.), так сразу большое значение начинает играть эффективность параллельной реализации операций этого вида.

При расширении исходной (последовательной однопроцессорной) модели сетки необходимо:

– создать модель разделенной сетки и разработать на ее основе распределенную модель;

– провести декомпозицию программного кода операций модификации сетки и реализовать их распределенную версию;

– распараллелить и оптимизировать операции модификации сетки.

Пусть задана грубая сетка, на которой установлены расчетные данные. Рассмотрим следующую последовательность шагов типичного сеточного алгоритма над большими объемами данных, предназначенного для запуска на многопроцессорных системах.

Шаг 1. Разделение сетки – грубая расчетная сетка разделяется на заданное число подсеток, равное, например, числу процессоров. Для реализации этого шага будем использовать исходную объектно-ориентированную модель сетки [1], которую необходимо расширить путем введения нового класса, задающего подсетки. Затем расширенная модель будет наполнена функциональностью, связанной с различными алгоритмами разделения сеток [2].

Шаг 2. Распределение сетки – подсетки размещаются на процессорах. Поддержка распределенных между различными вычислительными узлами объектов невозможна без коммуникационного промежуточного программного обеспечения [4], использование которого в объектно-ориентированном моделировании неизбежно приводит к понятиям клиент–сервер. Серверами в данном случае удобно представить подсетки. Их взаимодействиями управляет объект, задающий разделенную сетку, – клиент. Распараллеливание достигается путем асинхронного вызова со стороны клиента методов удаленных объектов подсеток.

Шаг 3. Дробление сетки – в соответствии с тем или иным алгоритмом подсетки измельчаются для проведения расчетов на полученной в результате сетке. Этот этап может неоднократно выполняться в ходе вычислений с адаптивным перестроением сетки. Алгоритмы дробления уже реализованы в исходной модели расчетной сетки. Перед распределенной реализацией их необходимо записать в терминах разделенной сетки, другими словами, декомпозировать код в соответствии с декомпозицией сетки. После этого алгоритмы распараллеливаются.

Шаг 4. Перераспределение сетки – перемещение сеточных объектов с одной подсетки/процессора на другие с целью выравнивания связанной с ними нагрузки того или иного вычисленного алгоритма. Для эффективного выполнения данного шага применительно к самому широкому классу сеточных алгоритмов требуется наличие программных методов, обеспечивающих различные стратегии размещения сеточных объектов. Для одних численных методов требуется пересылать сеточные объекты и все, связанные с ними расчетные данные, для других целесообразнее пересчитывать те или иные параметры заново, а иногда лучше избегать пересылки объектов, а передавать только некоторые их параметры.

2. Декомпозиция кода в модели разделенной сетки

Характерной особенностью созданной модели является то, что и сетка, и подсетка представляются одним и тем же классом. Данный подход позволяет рассматривать подобласти как самостоятельные сетки, что, в свою очередь, позволяет применять к подобластям те же алгоритмы, использовать их в тех же расчетах, что и исходный класс сетки без необходимости внесения изменений в код.

С помощью данной структуры можно декомпозировать код алгоритмов, реализованных в исходной объектно-ориентированной модели. Это предварительная декомпозиция программы до перехода к параллельным вычислениям на многопроцессорных системах. Декомпозиции подвергаются методы перемещения (используются на 2-м и 4-м шагах) и перестроения (используются на 3-м шаге) сеточных объектов.

3. Размещение сеточных объектов в распределенной модели

Разработанная модель распределенной расчетной сетки является расширением модели разделенной расчетной сетки. Новым здесь является введение объектов клиент–сервер для подсеток и всех видов сеточных объектов. Классы клиент–сервер подсетки обеспечивают удаленный доступ сетки к подсеткам, размещенным на других узлах вычислительной системы. Клиенты-серверы сеточных объектов предназначены для представления границы – это позволяет сохранить целостность данных – клиентские объекты обеспечивают доступ к граничным объектам, которые как серверы существуют в единственном экземпляре.

Основная проблема, решаемая в распределенной модели, – это обработка объектов, принадлежащих двум или более подсеткам (граничными объектами). Если подсетки будут получать копии граничных объектов, возникнет проблема синхронизации этих копий. В больших сетках, количество узлов в которых достигает пяти-шести порядков, синхронизация граничных объектов будет занимать значительное дополнительное время и ресурсы. Кроме того, огромная дополнительная ответственность ложится на программиста, ведь существует большое множество случаев, когда может потребоваться синхронизация копий объектов, и предусмотреть их все крайне сложно. Если же синхронизация будет производиться всякий раз, когда с объектом произошли какие-либо незначительные изменения, это приведет к огромному количеству вызовов от клиента к удаленному серверу и обратно, а также значительным дополнительным расходам памяти.

В модели распределенной сетки были созданы классы клиентов и серверов для граничных объектов (узлов, ребер, граней). Таким образом, исчезает необходимость хранения копий объектов в подсетках и исчезает проблема синхронизации. Экземпляры граничных объектов существуют в

единственном числе и хранятся в остове сетки. Остов содержит все объекты границ разделенной сетки. Взаимодействие с ними организуется посредством CORBA [5] с помощью соответствующих клиентов. Помимо граничных узлов существуют также граничные ребра и грани, для них также созданы соответствующие классы, исполняющие роль клиента и сервера.

Введение цепочки «клиент–сервер» для граничных объектов позволяет избежать перерасхода ресурсов и уменьшить время, затрачиваемое на расчеты. Безусловно, передача данных по сети будет замедлять работу, но потери времени в этом случае несравнимы с потерями во времени при проведении синхронизации между копиями объектов, находящихся в разных сетках.

Важнейшим методом, реализованным на уровне распределенной модели, является размещение сеточных объектов и создание заместителей для них на сетке и подсетках. Этот метод используется при реализации различных стратегий перераспределения объектов. По сути стратегия перераспределения есть решение вопроса, где нужно разместить сам объект, а где следует обращаться к этому объекту удаленно, посредством заместителей. Когда в подсетках граница перестраивается адаптивно к геометрии, сеточные объекты, принадлежащие границе области, целесообразно размещать в подсетках, оставляя заместителей в остове. Если же граница области не изменяется или изменяется несущественно, то размещать серверы граничных объектов можно в остове.

4. Асинхронное выполнение методов на подсетках

Распределенная модель позволяет привлечь большее число ресурсов, например кластеров ЭВМ. Суммарное же время выполнения будет даже больше времени выполнения задачи на отдельной машине за счет сетевых коммуникаций, которые добавляются в последовательный код. Распределенная модель составляет основу для распараллеливания вызовов отдельных методов. При параллельной работе затраты времени на решение задачи будут существенно уменьшаться.

Распараллеливание вызовов организуется с помощью технологии CORBA AMI [5], которая позволяет вызывать методы удаленных объектов асинхронно, не дожидаясь возвращения ими результата. Для превращения распределенной модели в параллельную требуется расширение класса клиента и создание дополнительного класса, который будет обрабатывать ответ, приходящий от сервера после завершения выполнения метода. Изменения необходимо вносить лишь на стороне клиента, что удобно, так как избавляет от необходимости переписывать больше строк кода.

При асинхронных вызовах операция может быть поделена на две части: запрос серверу и ожидание ответа клиентом. При ожидании клиентом результатов происходит приостановка всех обращений к серверу, что дает

ему возможность закончить работу и вернуть результат выполнения метода. Как только сервер готов отдать результат, он связывается с обработчиком ответа, который, в свою очередь, сообщает связанному с ним асинхронному клиенту о том, что операция была завершена, после чего клиент может продолжать работу с другими методами.

На данный момент распараллеливание реализовано для методов загрузки разделенной сетки из набора файлов, переноса объектов на удаленные серверы при делении и сохранение сетки вместе с подсетками, находящимися на удаленных серверах.

5. Перераспределение сеточных объектов

Динамическое перераспределение сеточных объектов применяется при балансировке нагрузки между вычислительными узлами. При перераспределении сеточных объектов, помимо загруженности вычислительных узлов, немаловажную роль играют затраты на выполнение непосредственно пересылки данных сеточных объектов. Выделим три стратегии перераспределения объектов [6]:

1. Перемещение сеточного объекта со всеми свойствами. Эта стратегия является наиболее затратной из всех рассматриваемых, поскольку необходимо переместить все данные, и выбирается в качестве базовой в построенной модели.
2. Перемещение сеточного объекта с созданием некоторых (или всех) из его свойств заново. Применяется в том случае, если восстановить данные свойств проще и быстрее, чем переслать их по коммуникационной сети. Для реализации этой стратегии у класса, реализующего конкретное свойство, в функции перемещения на подобласть необходимо, вместо реального перемещения, указать уничтожение свойства с созданием его на другой стороне и запустить его инициализацию.
3. Удаленный доступ через заместителя без реального перемещения объекта. Данная стратегия является самой экономной по времени перераспределения, но самой затратной в дальнейших вычислениях. Она может использоваться в том случае, когда пересылка данных всего объекта очень длительна и прогнозируется повторное перемещение объекта. В этом случае используется заместитель объекта, который создаётся автоматически при попытке доступа к объекту из другой подсетки.

Одним из методов решения задачи динамического перераспределения является последовательная обработка сеток, т.е. в определенный момент времени расчет и разделение происходят только на одной подсетке. В этом случае худшее, что можно ожидать, – некоторые объекты переместятся дважды, например, будучи граничными между подсетками, они на операции перераспределения одной подсетки станут внутренними, а на операции следующей подсетки – снова граничными, или наоборот. Но число та-

ких объектов будет достаточно небольшое, и по сравнению с общим объемом пересылок практически ничтожно.

Другой метод предназначен для параллельного перераспределения сеток. После того как становится известным расположение ячеек, рассылается информация в минимальном формате по всем подсеткам, тогда при определении расположения граней на каждом вычислительном узле будет находиться вся необходимая информация. После каждого следующего цикла определений расположения (для граней и для ребер) снова происходят рассылки. Когда определятся места расположения узлов, начнется фактическое перемещение объектов.

Заключение

Основная идея данной работы – построение и использование программной модели расчетной сетки в параллельных вычислениях. Для этого применялись следующие методы: декомпозиция методов перемещения и перестроения сеточных объектов исходной модели в соответствии с новой структурой данных, представляющей разделенную сетку в виде остова и набора подсеток; реализация подхода клиент–сервер с помощью промежуточного программного обеспечения CORBA при создании распределенной модели; распараллеливание на основе асинхронного выполнения методов в подсетках; определение методов размещения клиентов и серверов сеточных объектов для реализации различных стратегий перераспределения объектов.

Литература

1. Копысов С.П., Пономарев А.Б., Рычков В.Н. Открытое визуальное окружение для взаимодействия с геометрическими ядрами, генерации/перестроения/разделения сеток и построения расчетных моделей // Прикладная геометрия, построение расчетных сеток и высокопроизводительные вычисления / Ред. Ю.Г. Евтушенко, М.К. Керимов, В.А. Гаранжа. – М.: ВЦ РАН, 2004. – Т. 2. – С. 154–164.
2. Копысов С.П., Новиков А.К. Параллельные алгоритмы адаптивного перестроения и разделения неструктурированных сеток // Математическое моделирование. − 2002. Т. 14, № 9. − С. 91–96.
3. Копысов С.П., Красноперов И.В. Адаптивное решение задач в p-версии метода конечных элементов // Прикладная геометрия, построение расчетных сеток и высокопроизводительные вычисления / Ред. Ю.Г. Евтушенко, М.К. Керимов, В.А. Гаранжа. − М.: ВЦ РАН, 2004. − Т. 2. − С. 85−95.
4. Копысов С.П., Красноперов И.В., Рычков В.Н. Совместное использование систем промежуточного программного обеспечения CORBA и MPI // Программирование. − 2006. − № 4 (принята в печать).

5. Цимбал А. Технология CORBA для профессионалов. − СПб.: Питер, 2001. − 624 с.
6. Schmidt D.C. and Vinoski S. Programming Asynchronous Method Invocations with CORBA Messaging // C++ Report, SIGS. – 1999. − Vol. 11. № 2.
7. Копысов С.П., Красноперов И.В., Пономарев А.Б., Рычков В.Н. Модель расчетных неструктурированных сеток для параллельных распределенных вычислений // Математическое моделирование. − 2006 (принята в печать).

ПАРАЛЛЕЛЬНЫЙ ПРЯМО-ДВОЙСТВЕННЫЙ АЛГОРИТМ ОТСЕЧЕНИЙ ДЛЯ РЕШЕНИЯ СТРУКТУРИРОВАННЫХ ЛИНЕЙНЫХ ОПТИМИЗАЦИОННЫХ ЗАДАЧ

А.С. Величко

Институт автоматики и процессов управления ДВО РАН, Владивосток

Введение

Актуальность проблемы решения оптимизационных задач связана с усложнением формальных постановок количественных моделей исследования операций, математического моделирования, развитием новых моделей в экономике и финансовой математике. Зачастую структура матрицы ограничений экстремальной задачи обладает структурными особенностями, предоставляющими возможности для «крупнозернистого» представления задачи на относительно слабо связанные блоки меньшей размерности.

Алгоритмы решения больших (порядка 10^4 переменных/ограничений) и сверхбольших (порядка 10^5 – 10^6 переменных/ограничений) линейных экстремальных задач и систем линейных уравнений остаются важной частью арсенала математического программирования, исследования операций, и повышение их вычислительной эффективности имеет важное практическое значение. С распространением подходов, основанных на параллельных вычислениях, возникла проблема параллелизации симплексоподобных алгоритмов, все еще играющих важную роль в приложениях. Среди таковых надо особо отметить потребности в эффективном решении серий задач, являющихся небольшой модификацией друг друга, в чем современные симплексоподобные алгоритмы являются наиболее успешными. Подобные ситуации типичны, например, для приложений, в которых линейное программирование используется для решения внутренних вспомогательных задач типа метода линеаризации, итеративных проективных алгоритмов, методов негладкой оптимизации, целочисленного программирования и др.

Большой потенциал заключается, однако, в использовании параллельных вычислений для решения задач большой размерности со специальной, но часто встречающейся структурой. В первую очередь это относится к блочно-диагональной структуре матрицы ограничений и применению методов типа методов Данцига−Вульфа.

Постановка структурированной оптимизационной задачи

В работе [1] был предложен прямо-двойственный алгоритм, используемый для решения двублочных оптимизационных задач, и доказана его сходимость при достаточно общих предположениях. В качестве основной модели структурированной оптимизационной задачи рассмотрим двублочную проблему следующего вида:

$$\min_{z_A, z_B, x} c_A z_A + c_B z_B, \tag{1}$$

$$A_A z_A + B_A x \le d_A, \tag{2}$$

$$A_B z_B + B_B x \le d_B, \tag{3}$$

$$z_A, z_B \ge 0, \tag{4}$$

где z_A, z_B – векторы переменных задачи; x – вектор связывающих переменных; A_A, B_A, A_B, B_B – матрицы соответствующих размерностей.

При фиксированном x эта задача распадается на два независимых блока, что используется для развития декомпозиционного алгоритма двойственных отсечений.

Введя функции отклика для $M=\{A,B\}$

$$f_M(x) = \min_{z_M \ge 0} c_M z_M, A_M z_M \le d_M - B_M x, \tag{5}$$

получим эквивалентную для (1)−(4) задачу

$$\min_x f_A(x) + f_B(x). \tag{6}$$

Используя сопряженные функции $h_M(p) = \max_x \{px - f_M(x)\}$ задачу (6) легко переписать в терминах сопряженных функций:

$$\min_p h_A(-p) + h_B(p). \tag{7}$$

Такая эквивалентность задач (6) и (7) позволяет организовать эффективный процесс обмена координирующей информацией между прямыми и двойственными задачами линейного программирования в декомпозиционном подходе к решению задачи (1)−(4).

Разработка параллельного прямо-двойственного алгоритма отсечений

Суть алгоритма двойственных отсечений заключается в одновременном решении пары двойственных задач $\min_x f_A(x) + f^k(x)$ и $\min_p h^k(-p) + h_B(p)$. Внешние линейные аппроксимации для функций $f_B(x)$ и $h_A(p)$ имеют вид

$$f^k(x) = \max\{f^{k-1}(x), \bar{p}^k x - h_B(\bar{p}^k), p^k x - h_B(p^k)\},$$

$$h^k(p) = \max\{h^{k-1}(p), p\bar{x}^k - f_A(\bar{x}^k), px^{k+1} - f_A(x^{k+1})\},$$

а векторы $\bar{x}^k, \bar{p}^k$ являются субградиентами данных функций. Указанные задачи минимизации являются аппроксимированными вариантами задач

(6), (7). В результате их решения определяются субоптимальные векторы x^k, p^k. Вычисление значений функций $f_B(x)$, $h_A(p)$ и их субградиентов в точках x^k, $-p^k$ выступают подзадачами алгоритма. Решение подзадач на каждом шаге алгоритма позволяет строить новые линейные отсечения, которые уточняют кусочно-линейные аппроксимации $f^k(x)$ и $h^k(p)$. Блок-схема выполнения k-го шага МАДО представлена на рис. 1.

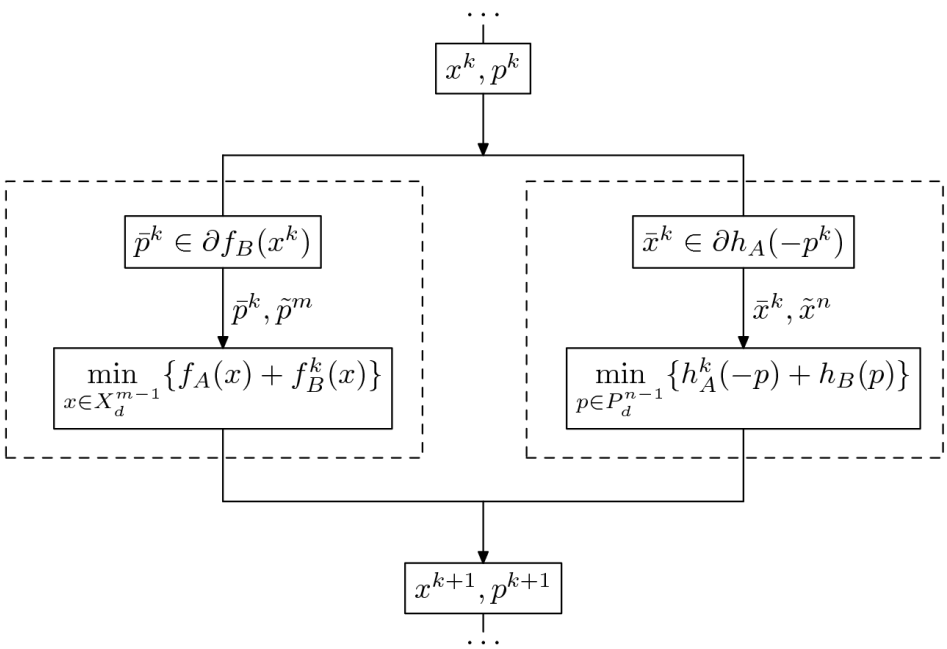


Рис. 1

В процессе выполнения алгоритма могут возникать некорректные задачи, в работе [2] показан способ построения дополнительных отсечений и модификации алгоритма в случаях недопустимости или неограниченности возникающих подзадач. Эта группа отсечений аппроксимирует множество определения функций $f_B(x)$ и $h_A(p)$. Множества

$$X_d^m = X_d^{m-1} \cup \{x : \tilde{p}^m x \le \tilde{g}^m\}, \quad P_d^n = P_d^{n-1} \cup \{p : \tilde{x}^m p \le 0\}$$

обновляются в случае возникновения неограниченных или недопустимых подзадач алгоритма, когда генерируемые алгоритмом векторы x^k, $-p^k$ не принадлежат области определения функций $f_B(x)$ и $h_A(p)$.

Возможны различные стратегии построения отсечений, они показаны на рис. 2, 3. После выполнения вычислений в каждом из блоков осуществляется обмен значениями x^{k+1} и p^{k+1} между процессами для вычислений на следующей итерации алгоритма. Более подробно вычислительные аспекты использования таких прямо-двойственных отсечений описаны, в частности, в работах [1,2].

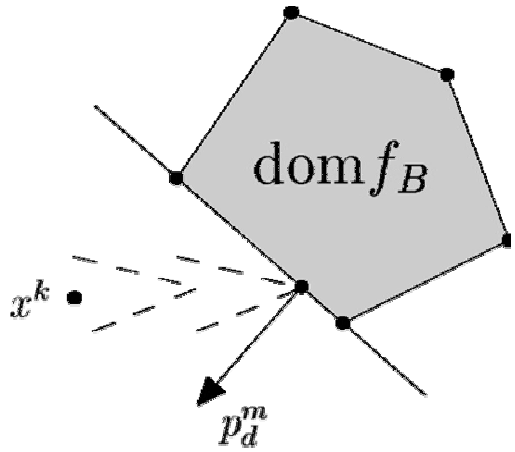


Рис. 2

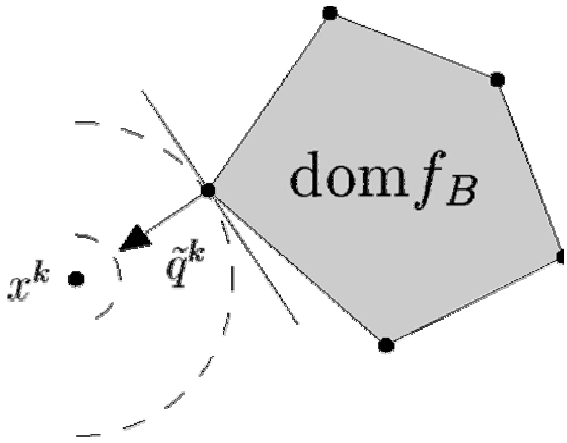


Рис. 3

Важными вычислительными задачами алгоритма являются вычисления субградиентов функций $f_B(x)$ и $h_A(p)$. Пусть $f_B(x)$ определена соотношением (5). Тогда для некоторого фиксированного вектора $x*$ имеем

$$f_B(x) = \min_{z_M \geq 0} c_B z_B, A_B z_B \leq d_B - B_B x, x = x*. \tag{8}$$

Из стандартной теории двойственности линейного программирования и теории выпуклого анализа легко получить следующее утверждение.

Утверждение. *Пусть для заданного $x*$ задача (8) разрешима, $p*$ является вектором оптимальных двойственных переменных к ограничению $x = x*$ в задаче (8). Тогда $p* \in \partial f_B(x*)$, и обратно, любой вектор $p* \in \partial f_B(x*)$ является вектором оптимальных двойственных переменных, соответствующих ограничению $x = x*$ в задаче (8).*

Направлением дополнительного повышения численной эффективности алгоритма является использование е-отсечений, основанных на приближенном решении подзадач алгоритма. Для этого необходимо уметь вычислять е-субградиенты кусочно-линейных функций отклика. Важным для приложений является вопрос использования минимальной доступной прямо-двойственной информации, получаемой в процессе приближенного итеративного решения подзадач.

В приведенных выше формулировках функции $f_B(x)$ и $h_A(p)$ являются выпуклыми кусочно-линейными функциями, в этих условиях удается доказать сходимость алгоритма с использованием приближенных решений подзадач.

Теорема. *Пусть $f_B(x)$ и $h_A(p)$, определенные в (5), являются конечными, выпуклыми и кусочно-линейными, задачи поиска субградиентов функций $f_B(x)$ и $h_A(p)$ решаются приближенно с некоторой точностью ε_k и*

$\lim_{k\to\infty} \varepsilon_k = 0$. *Последовательности* $\{x^k\}$, $\{p^k\}$, *генерируемые алгоритмом, ограничены. Тогда все предельные точки последовательностей* $\{x^k\}$, $\{p^k\}$ *являются решениями задач (6), (7) соответственно.*

Теорема демонстрирует, что вместо использования точных значений субградиентов функций отклика на каждом шаге алгоритма можно вычислять последовательность их е-субградиентов, сходящихся по мере выполнения алгоритма к точным субградиентам.

Задача двушагового стохастического программирования в блочной форме

Рассмотрим процесс декомпозиции оптимизационной задачи на блоки ограничений на примере задачи двушагового стохастического программирования. В линейных оптимизационных задачах стохастического программирования, возникающих в ситуации неопределенности, когда переменные модели являются случайными величинами, возникает следующая математическая постановка вычисления оптимального вектора x:

$$\min_{x\geq 0} c(w)x, S(w)x = h(w),$$

где w – элементарное событие на заданном пространстве событий Ω.

Очевидно, что решение x зависит от реализации определенного события *w*. Однако если решение этой задачи должно быть определено *a priori*, вряд ли найдется вектор x, удовлетворяющий ограничениям задачи для всех событий $w \in \Omega$. Вектор $\delta(x,w) = h(w) - S(w)x$ характеризует степень нарушения ограничений исходной задачи при выборе какого-то x. В двушаговой модели стохастического программирования предполагается, что на первом шаге мы знаем лишь множество событий Ω, на втором шаге мы получаем полную информацию о наступившем событии *w*. В таком случае оптимальный выбор x на первом шаге может проводиться при одновременной минимизации среднего ожидаемого значения величины $c(w)x$ и некоторого (линейного) функционала от вектора $\delta(x,w)$. Выбирая в качестве $\delta(x,w)$ сумму абсолютных значений компонент, запишем исходную задачу в виде

$$\min_{x\geq 0}[\mathrm{M}\{c(w)x\} + \min_{y^+,y^-} \mathrm{M}\{ey^+ + ey^-, y^+ - y^- = h(w)\text{-}S(w)x\}, y^+ \geq 0, y^- \geq 0],$$

где *M* – символ математического ожидания. В случае, когда все множество событий Ω содержит всего два события w_1, w_2, вероятность наступления которых равны соответственно $p_1, p_2 = 1 - p_1$, последняя задача имеет вид

$$\min_{y^+,y^-,x} [(p_1c_1 + p_2c_2)x + p_1(ey_1^+ + ey_1^-) + p_2(ey_2^+ + ey_2^-)],$$

$$y_1^+ - y_1^- + S^1x = h^1,\ y_2^+ - y_2^- + S^2x = h^2,\ x, y_i^+, y_i^- \geq 0,$$

где e – вектор из единиц, $S^i = S(w_i)$, $h^i = h(w_i)$, *i=1,2*.

Ясно, что система ограничений данной задачи является блочной вида (1)–(4). Однако вектор связывающих переменных x в данной задаче входит в минимизируемый функционал.

Обозначим новый вектор связывающих переменных за $\tilde{x}$. Добавим ограничение $x = \tilde{x}$ и агрегируем его, например, с ограничениями $y_1^+ - y_1^- + S^1 x = h^1$. Дальнейшее агрегирование вектора x с векторами y^+, y^- позволяет исключить вхождение вектора x в функционал получаемой задачи, сохраняя двублочность ограничений.

В результате такой агрегации получим структурированную двублочную задачу

$$\min_{\tilde{y}_1, \tilde{y}_2, \tilde{x}} \tilde{c}_1 \tilde{y}_1 + \tilde{c}_2 \tilde{y}_2,$$

$$A_A \tilde{y}_1 + B_A \tilde{x} \le \tilde{h}_1,$$

$$A_B \tilde{y}_2 + S_2 \tilde{x} \le h_2,$$

$$\tilde{x}, \tilde{y}_1, \tilde{y}_2 \ge 0,$$

где $\tilde{x}$ – новый вектор связывающих переменных.

Реализация параллельного прямо-двойственного алгоритма отсечений и численные эксперименты

Для программной реализации параллельного алгоритма использована технология параллельных вычислений стандарта MPI реализации LAM. Вычисления организованы в виде процессов, обменивающихся сообщениями стандарта MPI. Разработанная программа используется на многопроцессорных вычислительных комплексах МВС-1000/16-17 в составе Центра коллективного пользования «Дальневосточный вычислительный ресурс» http://www.dvo.ru/bbc.

Для реализации операций линейной алгебры используется свободно распространяемая библиотека meschach. Для решения подзадач используется симплекс-метод с одновременным определением двойственных переменных и факторизацией базисной матрицы для ее более эффективного обращения. Для дополнительного сокращения вычислений применяется процедура «досчета», когда оптимальное решение задачи, полученное на k-м шаге алгоритма, используется в качестве допустимого решения для решения этой же задачи на (k+1)-м шаге.

В таблице представлена статистика тестовых примеров различной размерности для задачи двушагового стохастического программирования, использующих случайно генерируемые данные. «Количество ограничений» – общее количество ограничений задачи, столбец «Время 1» – время выполнения последовательной версии прямо-двойственного алгоритма в секундах, «Время 2» – время выполнения параллельной реализации алгоритма в секундах. В столбце «Ускорение» показано отношение времени

выполнения последовательной параллельной реализаций алгоритма, в последнем столбце «Выигрыш, %» показано относительное уменьшение времени счета при использовании параллельного алгоритма по сравнению с его последовательным вариантом.

Тест	Количество ограничений	Время 1 (с)	Время 2 (с)	Ускорение	Выигрыш, %
1	209	13,12	9,15	1,43	30,3
2	409	257,54	234,16	1,1	9,1
3	609	1680,79	1226,4	1,37	27,0
4	809	6697,7	5044,71	1,33	24,7
5	1009	17977,52	11975,57	1,50	33,4
6	1209	29252,07	27147,8	1,08	7,2
7	1409	54066,36	41857,47	1,29	22,6
8	1609	111173,05	82079,81	1,35	26,2
9	1809	198491,13	157503,63	1,26	20,6
10	2009	225800,69	200270,27	1,13	11,3

При использовании двух параллельно выполняемых процессов максимальное теоретическое значение коэффициента ускорения равно 2, когда время выполнения процессов на каждом шаге алгоритма одинаково при относительно небольшом объеме обмена данными между процессами. Анализ времени выполнения каждого из блоков алгоритма показывает, что происходит простой второго блока параллельного алгоритма до момента обмена значениями x^{k+1}, p^{k+1} между процессами и перехода на следующий шаг алгоритма, поскольку на решение задач первого блока затрачивается относительно большее время. Такой простой происходит в силу трудоемкости задачи поиска точного субградиента функции $f_B(x)$ в точке из-за невозможности использования описанной выше стратегии “досчета” в этой задаче.

На рис. 4 представлены результаты выполнения тестов для задачи двушагового стохастического программирования на случайно генерируемых данных. Алгоритм демонстрирует линейный характер сходимости, выигрыш от использования параллелизма по времени выполнения алгоритма составляет в большинстве тестов 20–25%.

Заключение

В работе рассмотрен способ организации параллельных вычислений для модифицированного алгоритма двойственных отсечений, разработанного в [1,2], применяемого для эффективной прямо-двойственной декомпозиции структурированных задач линейного программирования. Рассматривается постановка задачи двушагового стохастического программи-

рования, показано, что она может быть представлена в классе структурированных оптимизационных задач. На рис. 4 $2m+Nx$ – число ограничений задачи. Масштаб шкалы логарифмический по обеим осям. Линия обычной толщины соответствует последовательному, а жирная линия – параллельному алгоритму прямо-двойственных отсечений.

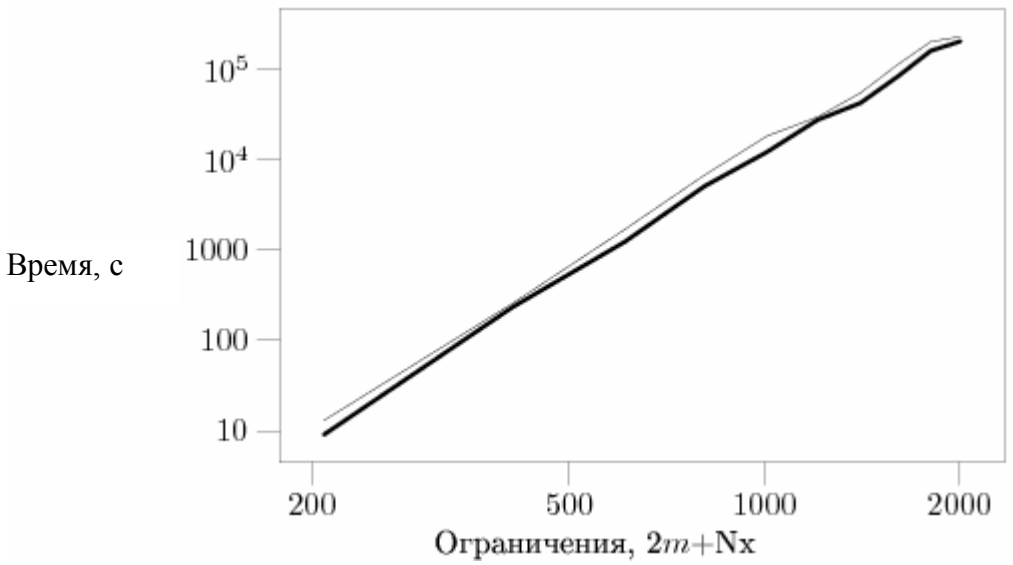


Рис. 4

Для решения подзадач алгоритма был реализован алгоритм симплекс-метода с одновременным вычислением двойственных переменных. Использована стратегия «досчета» допустимого решения подзадач, в качестве которого используется оптимальный базис этой подзадачи на предыдущем шаге алгоритма. Применение данной стратегии возможно для всего класса методов отсечений и обусловлено их спецификой. Использование процедуры LU-разложения для обращения базисной матрицы позволяет дополнительно сократить вычислительные затраты симплекс-метода.

Проведены вычислительные эксперименты с параллельным и последовательным версиями алгоритма на примере задачи двушагового стохастического программирования большой размерности со случайно генерируемыми данными. Алгоритмы демонстрируют линейную скорость сходимости, что наблюдалось авторами в их применении к задачам большой размерности в жестких экстремальных задачах с тривиальным носителем [2] и задаче финансовой математики о репликации портфеля рыночных активов [3]. В этих различных по структуре задачах используемый алгоритм показал свою устойчивость и эффективность, трудоемкость которого определяется в основном количеством связывающих переменных задачи.

Литература

1. Нурминский Е.А. Численные методы выпуклой оптимизации. – М.: Наука, 1991. − 165 с.
2. Величко А.С., Нурминский Е.А. Опыт декомпозиции метода конечных элементов с использованием теории структурированных оптимизационных задач // Электронный журнал «Исследовано в России». – 2002. № 113. – С. 1237–1256. – http://zhurnal.ape.relarn.ru/articles/2002/113.pdf
3. Величко А.С., Нурминский Е.А. Прямо-двойственная декомпозиция задачи о репликации портфеля рыночных активов // Автоматика и телемеханика. – 2004. – № 2. – С. 170–178.

ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ ДЛЯ СПЛАЙНОВОЙ СИСТЕМЫ ОБЫКНОВЕННЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ

В. Н. Берцун

Томский государственный университет

Сплайны широко используются для приближения функций, автоматизации проектирования или изготовления изделий сложной геометрической формы. Такие функции успешно применяются в задачах гидродинамики и тепломассообмена, а также для конструирования экономичных разностных схем решения многомерных краевых задач. Отличие разностных схем сплайновой интерполяции от традиционных состоит в том, что имеется согласованная со схемой достаточно простая интерполяционная функция. Это упрощает проблему перестроения пространственной сетки в процессе счета (если в этом возникает необходимость), дает возможность гладкого восстановления решения и его производных по сеточной функции в любой точке расчетной области, упрощает визуализацию расчетов. Используя интерполяционные кубические сплайны с априорно заданными свойствами решения (гладкость, разрывы производных, монотонность и др.), к которым относится и итерационно-интерполяционный метод (ИИМ) [1], можно получать консервативные разностные схемы для решения краевых задач, применяя гибкую систему независимых аппроксимаций.

Появление многопроцессорных вычислительных систем с распределенной памятью стимулирует работы по математическому моделированию сложных объектов, созданию параллельных вычислительных алгоритмов на основе методов декомпозиции расчетной области и сплайновых восстановлений многомерных сеточных функций.

Рассмотрим краевую задачу

$$\frac{\partial u}{\partial t} = a^2 \frac{\partial^2 u}{\partial x^2} + f(u,x,t),\ u(x,0) = \varphi(x),\ \{0 \le x \le 1, 0 < t \le T\},$$

$$\lambda \frac{\partial u}{\partial x} = \alpha(u - T), x = 0; \lambda \frac{\partial u}{\partial x} = \alpha(T - u), x = 1,$$

где a^2, α, T, λ – заданные константы.

Сплайновая система обыкновенных дифференциальных уравнений ИИМ первого приближения для такой задачи имеет вид [1]

$$\delta\left(2V_0' + V_1'\right) = \frac{\lambda\ (V_1 - V_0)}{h}\ - \alpha\ (V_0 - T_0) + \delta\mu\left(2f_0 + f_1\right),$$

$$V''_{i-1} + 4V''_i + V''_{i+1} = 6a^2 \Lambda V_i + \mu\,(f_{i-1} + 4f_i + f_{i+1}),\ i = \overline{1, N-1}; \qquad (1)$$

$$\delta\,(2V_N + V_{N-1}) = \frac{\lambda\,(V_{N-1} - V_N)}{h} + \alpha_1 (T_1 - V_N) + \delta\mu\,(2f_N + f_{N-1}),$$

$$V_i(0) = \varphi_i,\ i = \overline{0, N},\ \alpha,\ \alpha_1 \geq 0,\ \delta = \frac{\lambda h}{6a^2} > 0.$$

Эту систему можно переписать в нормальной векторной форме

$$\frac{d\bar{u}}{dt} = A^{-1}B\bar{u} + \bar{f} + A^{-1}\bar{b}\ ,\ \bar{u}(0) = \bar{\varphi}, \qquad (2)$$

где A, B – трехдиагональные квадратные матрицы $(N+1)$x$(N+1)$

$$A = \begin{pmatrix} 2 & 1 & 0 & \cdots & 0 \\ 1 & 4 & 1 & & 0 \\ \ddots & \ddots & \ddots & & \ddots \\ 0 & 1 & 4 & 1 & \\ 0 & \cdots & 0 & 1 & 2 \end{pmatrix},\quad B = \begin{pmatrix} -\nu_1 & \nu_2 & 0 & \cdots & 0 \\ \gamma & -2\gamma & \gamma & & 0 \\ \ddots & \ddots & \ddots & & \ddots \\ 0 & \gamma & -2\gamma & \gamma & \\ 0 & \cdots & 0 & \nu_2 & -\nu_3 \end{pmatrix},$$

$$\bar{b} = \begin{pmatrix} \alpha & T_0/\delta \\ 0 & \\ \vdots & \\ 0 & \\ \alpha_1 & T_1/\delta \end{pmatrix},\ \bar{f} = \begin{pmatrix} f_0 \\ \vdots \\ \vdots \\ \vdots \\ f_N \end{pmatrix},\ \bar{V} = \begin{pmatrix} V_0 \\ \vdots \\ \vdots \\ \vdots \\ V_N \end{pmatrix},\ \bar{\varphi} = \begin{pmatrix} \varphi_0 \\ \vdots \\ \vdots \\ \vdots \\ \varphi_N \end{pmatrix},$$

$$\nu_1 = \left(\frac{\lambda}{h} + \alpha\right)/\delta\ ,\quad \nu_2 = \lambda\,(h\,\delta),$$

$$\nu_3 = \left(\frac{\lambda}{h} + \alpha_1\right)/\delta\ ,\quad \gamma = 6\,a^2/h^2.$$

Численное решение системы ОДУ ИИМ (1) можно найти методом Рунге–Кутта с итерациями. Решение системы (2) находится без итераций, но требует вычисления элементов матрицы $C=A^{-1}B$.

Так как трехдиагональная матрица A со строгим диагональным преобладанием, то элементы ее обратной матрицы быстро убывают с удалением от главной диагонали.

Например, в таблице приведена матрица A^{-1} для $N=4$.

i,j	0	1	2	3	4
0	**0,577**	**–0,155**	0,042	–0,012	0,006
1	**-0,155**	**0,31**	**–0,083**	0,024	–0,012
2	0,042	**–0,083**	**0,292**	**–0,083**	0,042
3	–0,012	0,024	**–0,083**	**0,31**	**0,155**
4	0,006	–0,012	0,042	**–0,155**	**0,577**

Для определения элементов обратной матрицы A^{-1} в (2) используем идею метода Гаусса [2] . Введем обозначения:

$$A = \begin{pmatrix} a_0 & b_0 & 0\cdots 0 \\ c_1 & a_1 & b_1 & 0 \\ \ddots & \ddots & \ddots & \\ 0 & c_{N-1} & a_{N-1} & b_{N-1} \\ 0\cdots 0 & & c_N & a_N \end{pmatrix}, \qquad A^{-1} = \begin{pmatrix} d_{00} & \cdots & d_{0N} \\ d_{10} & \cdots & d_{1N} \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ d_{N0} & \cdots & d_{NN} \end{pmatrix},$$

$$a_i - (b_i + c_{\,i}) = \Delta_i > 0, \quad i = \overline{0, N}.$$

Так как $A^{-1}A = E$, то элементы m-го столбца обратной матрицы являются решением системы линейных уравнений с трехдиагональной матрицей

$$\begin{aligned} & a_0\, d_{0m} + b_0\, d_{1m} = 0; \\ & c_k\, d_{k-1,m} + a_k\, d_{km} + b_k\, d_{k+1,m} = 0,\ k = \overline{1, m-1}; \\ & c_m\, d_{m-1,m} + a_m\, d_{mm} + b_m\, d_{m+1,m} = 1; \\ & c_k\, d_{k-1,m} + a_k\, d_{km} + b_k\, d_{k+1,m} = 0, \quad k = \overline{m+1, N-1}; \\ & c_N\, d_{N-1,m} + a_N\, d_{Nm} = 0, \quad m = \overline{0, N}\ . \end{aligned} \tag{3}$$

Учитывая соотношения метода встречных прогонок, для системы (3) получим

$$d_{im} = \eta_i\, d_{i+1,m} + \lambda_i, \quad i = \overline{0, N-1}; \tag{4}$$

$$d_{im} = \overline{\eta}_i\, d_{i-1,m} + \overline{\lambda}_i, \quad i = N, N-1, \ldots, 1; \tag{5}$$

$$\eta_0 = -\frac{b_0}{a_0}, \quad \eta_i = \frac{-b_i}{a_i + c_i\,\eta_i}, \quad \lambda_i = 0, \quad i = \overline{0, m-1};$$

$$\overline{\eta}_N = -\frac{c_N}{a_N}, \quad \eta_i = \frac{-c_i}{a_i + b_i\,\overline{\eta}_{i+1}}, \quad \overline{\lambda}_i = 0, \quad i = N, N-1, \ldots, m+1.$$

Исключив из m-го уравнения системы (3) $d_{m-1,m}$ и $d_{m+1,m}$ с помощью формул (4) и (5), определим диагональный элемент

$$d_{i\,m} = (a_m + c_m\eta_{m-1} + b_m\overline{\eta}_{m+1})^{-1}, \tag{6}$$

а затем и все остальные компоненты m-го столбца матрицы A^{-1}.

Для упрощения правой части системы (2), учитывая локальные свойства кубических сплайнов и оценку для элементов обратной матрицы

$$\left|d_{ij}\right| \le \frac{1}{a_j}\left(\max_{0\le k\le N}\left(\frac{a_k}{\Delta_k}\right)\right)\left(1 - \min_{0\le k\le N}\left(\frac{\Delta_k}{a_k}\right)\right)^{|i-j|} < 1; \quad i, j = \overline{0, N},$$

вычислим точно только трехдиагональную часть матрицы A^{-1}. Тогда в каждой системе уравнений (3), после определения диагонального элемента по формуле (6), вычисляются только значения $d_{i-1,m}$, $d_{i+1,m}$, а остальные эле-

менты считаются нулевыми [3]. Обозначив такую матрицу через $A_3^{-1} = \left[d_{ij}\right]_0^N$, получим приближенное представление системы (2) в нормальной форме

$$\dot{\bar{V}} = C\bar{V} + \bar{f} + A_3^{-1}\bar{b},\ \bar{V}(0) = \bar{\varphi},\ C = A_3^{-1}B. \qquad (7)$$

Так как матрицы $A_3^{-1} = [d_{ij}]_0^N$, $B = [b_{ij}]_0^N$ трехдиагональные, то элементы пятидиагональной матрицы C можно вычислить по формулам:

$$c_{00} = \sum_{k=0}^{1} d_{0k} b_{k0},\ c_{01} = \sum_{k=0}^{1} d_{0k} b_{k1},\ c_{02} = d_{01} b_{12},$$

$$c_{0i} = 0,\ i = \overline{3, N},$$

$$c_{10} = \sum_{k=0}^{1} d_{1k} b_{k0},\ c_{11} = \sum_{k=0}^{2} d_{1k} b_{k1},\ c_{12} = \sum_{k=1}^{2} d_{1k} b_{k2},$$

$$c_{13} = d_{12} b_{23},\ c_{1i} = 0,\ i = \overline{4, N},$$

$$c_{20} = d_{21} b_{10},\ c_{21} = \sum_{k=1}^{2} d_{2k} b_{k1},\ c_{22} = \sum_{k=1}^{3} d_{2k} b_{k2},\ c_{23} = \sum_{k=2}^{3} d_{2k} b_{k3},$$

$$c_{24} = d_{23} b_{34},$$

$$c_{2i} = 0,\ i = \overline{5, N};\ c_{30} = 0,\ c_{31} = d_{32} b_{21},\ c_{32} = \sum_{k=2}^{3} d_{3k} b_{k2}, \triangleleft$$

$$c_{33} = \sum_{k=2}^{4} d_{3k} b_{k3},\ c_{34} = \sum_{k=3}^{4} d_{3k} b_{k4},\ c_{35} = d_{34} b_{45},$$

$$c_{3i} = 0,\ i = \overline{6, N};\ \ldots,$$

$$c_{N-3\,i} = 0,\ \ i = \overline{0, N-6},\ c_{N-3\,N-5} = d_{N-3\,N-4} b_{N-4\,N-5},$$

$$c_{N-3\,N-4} = \sum_{k=N-3}^{N-4} d_{N-3\,k} b_{k\,N-4},\ c_{N-3N-3} = \sum_{k=N-4}^{N-2} d_{N-3\,k} b_{k\,N-3},$$

$$c_{N-3\,N-2} = \sum_{k=N-3}^{N-2} d_{N-3\,k} b_{k\,N-2},\ \ c_{N-3\,N-1} = d_{N-3\,N-2} b_{N-2\,N-1},$$

$$c_{N-3\,N} = 0;\ c_{N-2\,i} = 0,\ i = \overline{0, N-5},\ c_{N-2\,N-4} = d_{N-2\,N-3} b_{N-3\,N-4},$$

$$c_{N-2\,N-3} = \sum_{k=N-3}^{N-2} d_{N-2\,k} b_{k\,N-3},\ c_{N-2N-2} = \sum_{k=N-3}^{N-1} d_{N-2\,k} b_{k\,N-2},$$

$$c_{N-2\,N-1} = \sum_{k=N-2}^{N-1} d_{N-2\,k} b_{k\,N-1},\ \ c_{N-2\,N} = d_{N-2\,N-1} b_{N-1\,N},$$

$$c_{N-1\,i} = 0\ ,\ i = \overline{0, N-4},\ \ c_{N-1\,N-3} = d_{N-1\,N-2} b_{N-2\,N-3},$$

$$c_{N-1\,N-2} = \sum_{k=N-2}^{N-1} d_{N-1\,k} b_{k\,N-2},\ c_{N-1N-1} = \sum_{k=N-2}^{N} d_{N-1\,k} b_{k\,N-1},$$

$$c_{N-1N} = \sum_{k=N-1}^{N} d_{N-1\,k} b_{k\,N},\ \ c_{N\,i} = 0,\ \ i = \overline{0, N-3},$$

$$c_{N\,N-2} = d_{N\,N-1} b_{N-1\,N-2}, \quad c_{N\,N-1} = \sum_{k=N-1}^{N} d_{N\,k} b_{k\,N-1},$$

$$c_{N\,N} = \sum_{k=N-1}^{N} d_{N\,k} b_{k\,N}.$$

Если C – симметричная матрица, то достаточно вычислить только верхнюю треугольную часть этой матрицы.

Решение системы ОДУ (7) можно осуществить методом Рунге–Кутта на многопроцессорной вычислительной системе с распределенной памятью [4–9]. В этом случае необходимая точность по пространственной переменной обеспечивается выбором достаточно большого числа промежутков N, а заданную точность по времени можно обеспечить применением расчетных формул соответствующей точности. При этом для любого $N \geq 5$ число обменов между соседними процессорами при вычислении правых частей системы (7) на каждом шаге по времени t не превосходит 2. Если функция $f(u, x, t)$ нелинейная, то для решения системы (7) можно использовать принцип расщепления по физическим процессам.

Для пошагового контроля точности решения задачи Коши можно использовать вложенные формулы Рунге – Кутта (ВФРК) [7] . Например, для линейной системы ОДУ

$$\dot{\bar{V}} = C\bar{V} \quad , \quad \bar{V}(0) = \bar{\varphi}, \quad t \in (0,\ T].$$

Мерсоном была предложена схема

$$u^{n+1} = u^n + \tau \left(\mathrm{k}_1^{\mathrm{n}} - 3\,\mathrm{k}_3^{\mathrm{n}} + 4k_4^n\right)/2 \ , \ \mathrm{n} = \overline{0, M-1}\ , \tag{8}$$

$$v^{n+1} = u^n + \tau \left(\mathrm{k}_1^{\mathrm{n}} + 4\,\mathrm{k}_4^{\mathrm{n}} + \mathrm{k}_5^{\mathrm{n}}\right)/6 \ , \ \mathrm{n} = \overline{0, M-1}\ . \tag{9}$$

где τ – шаг по времени, $t_n = n\,\tau$,

$$k_1^n = \left[\mathrm{C}\,\mathrm{u}^{\mathrm{n}}\right], \quad \mathrm{k}_2^{\mathrm{n}} = \left[\mathrm{C}\left(\mathrm{u}^{\mathrm{n}} + \tau\ \ \mathrm{k}_1^{\mathrm{n}}/3\right)\right],$$

$$k_3^n = \left[\mathrm{C}\left(\mathrm{u}^{\mathrm{n}} + \tau\,\mathrm{k}_1^{\mathrm{n}}/6 + \tau\,\mathrm{k}_2^{\mathrm{n}}/6\right)\right], \ \mathrm{k}_4^{\mathrm{n}} = \left[\mathrm{C}\left(\mathrm{u}^{\mathrm{n}} + \tau\,\mathrm{k}_1^{\mathrm{n}}/8 + 3\tau\,\mathrm{k}_3^{\mathrm{n}}/8\right)\right],$$

$$\mathrm{k}_5^{\mathrm{n}} = \left[\mathrm{C}\left(\mathrm{u}^{\mathrm{n}} + \tau\,\mathrm{k}_1^{\mathrm{n}}/2 - 3\tau\,\mathrm{k}_3^{\mathrm{n}}/2 + 2\tau k_4^n\right)\right].$$

Расчетная формула (8) имеет четвертый порядок точности, а для контроля точности на каждом шаге вычисляют

$$\delta^{n+1} = \frac{\max}{i = \overline{1, N}} \left| u_i^{n+1} - v_i^{n+1} \right|,$$

где формула (9) имеет пятый порядок точности.

Для повышения эффективности параллельного алгоритма запишем (8), (9) в операторной форме

$$u^{n+1} = B_1 u^n \ , \quad v = B_2 u^n \ , \quad \delta^{n+1} = h^2 C^4 u^n / 4. \tag{10}$$

Здесь

$$B_1 = [E + \tau C + \tau^2 C^2/2 + \tau^3 C^3/6 + \tau^4 C^4/24],$$

$$B_2 = [E + \tau C + \tau^2 C^2 / 2 + \tau^3 C^3 / 6].$$

В этом случае предварительно с помощью параллельного алгоритма умножения матриц вычисляется P строчных блоков матриц B_1 и B_2 , а каждый блок хранится в i-м процессоре: $i=0,1,\ldots,P-1$.

Литература

1. Берцун В.Н. Сплайны сеточных функций. – Томск: Изд-во Том. ун-та, 2002. – 124 с.
2. Самарский А.А., Николаев Е.С. Методы решения сеточных уравнений. – М.:Наука, 1978. – 592 с.
3. Ильин В.П. Методы неполной факторизации для решения алгебраических систем. – М.: Физматлит, 1995. – 288 с.
4. Хайрер Э., Нерсетт С., Ваннер Г. Решение обыкновенных дифференциальных уравнений. Нежесткие задачи. – М.: Мир, 1990. – 512 с.
5. Арушанян О. Б., Залеткин С. Ф. Численное решение обыкновенных дифференциальных уравнений на Фортране. – М.: Изд-во МГУ, 1990. – 336 с.
6. Воеводин В. В., Воеводин Вл. В. Параллельные вычисления. – СПб.: БХВ-Петербург, 2002. – 608 с.
7. Фельдман Л. П. Параллельные алгоритмы численного решения систем линейных обыкновенных дифференциальных уравнений // Математическое моделирование. – 2000. – Т. 12, №6. – С. 15–20.
8. Фельдман Л. П., Назарова И. А. Эффективность параллельных алгоритмов вложенных методов Рунге–Кутта при моделировании сложных динамических систем // Высокопроизводительные параллельные вычисления на кластерных системах: Материалы Второго Международного научно-практического семинара. – Нижний Новгород: Изд-во ННГУ им. Н.И. Лобачевского, 2002. – 351 с.
9. Артемьев С.С. Численные методы решения задачи Коши для систем обыкновенных и стохастических дифференциальных уравнений. – Новосибирск: Изд-во ВЦ СО РАН, 1993. – 156 с.
10. Оран Э., Борис Дж. Численное моделирование реагирующих потоков. – М.: Мир,1990. – 660 с.
11. Вержбицкий В. М. Численные методы. – М.: Высшая школа, 2001. – 382 с.

ПАРАЛЛЕЛЬНАЯ РЕАЛИЗАЦИЯ ЯВНОЙ РАЗНОСТНОЙ СХЕМЫ ДЛЯ РАСЧЕТА ДВУХФАЗНЫХ ТЕЧЕНИЙ В КАНАЛАХ

Л.Л. Миньков, К.В. Озерская

Томский государственный университет

Рассматривается течение многофазной среды в плоском канале постоянного сечения, с закрытым торцом (рис. 1), с боковой поверхности которого осуществляется приход смеси газа и частиц с известной функцией распределения по размеру.

Требуется найти стационарное распределение параметров двухфазной смеси по каналу. Для решения таких задач используется метод установления. Предполагается, что решение соответствующей нестационарной задачи сходится к стационарному, если невязка решения падает на несколько порядков.

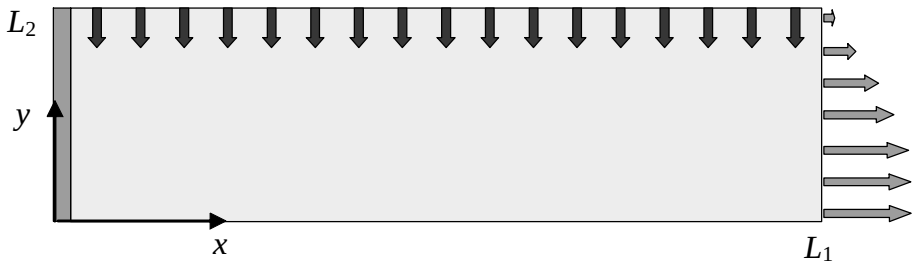


Рис. 1. Область решения

Определяющая система уравнений

Система уравнений, описывающая течение смеси газа и полидисперсных частиц в рамках многоскоростного и многотемпературного континуума [1], имеет вид

$$\oint_S \mathbf{U}dxdy + \mathbf{F}dtdy + \mathbf{G}dtdx = \iint_D \mathbf{E}dtdxdy, \qquad (1)$$

$$0 < x < L_1,\ 0 < y < L_2,\ t > 0,$$

$$\mathbf{U}=\begin{bmatrix}\rho\\ \rho U\\ \rho V\\ \rho E\\ \rho_s\\ \rho_s U_s\\ \rho_s V_s\\ \rho_s T_s\end{bmatrix};\quad \mathbf{F}=\begin{bmatrix}\rho U\\ \rho U^2+P\\ \rho UV\\ \rho UH\\ \rho_s U_s\\ \rho_s U_s^2\\ \rho_s V_s U_s\\ \rho_s T_s U_s\end{bmatrix};\quad \mathbf{G}=\begin{bmatrix}\rho V\\ \rho UV\\ \rho V^2+P\\ \rho VH\\ \rho_s V_s\\ \rho_s V_s U_s\\ \rho_s V_s^2\\ \rho_s T_s V_s\end{bmatrix};\quad \mathbf{E}=\begin{bmatrix}0\\ -\sum_{s=1}^{N}\rho_s f_{x,s}\\ -\sum_{s=1}^{N}\rho_s f_{y,s}\\ -\sum_{s=1}^{N}\rho_s\left(\vec{V}_s\vec{f}_s+q_s\right)\\ 0\\ \rho_s f_{x,s}\\ \rho_s f_{y,s}\\ \rho_s q_s/C_B\end{bmatrix},$$

где $\vec{f}_s=\varphi_{1,s}(\vec{V}-\vec{V}_s)$ и $q_s=\varphi_{2,s}(T-T_s)$ – соответственно, сила и тепловой поток, действующие на частицу со стороны газа; ρ – плотность; U, V – составляющие вектора скорости вдоль x и y осей; P – давление; E – полная энергия, $E=e+u^2/2$; e – внутренняя энергия газа, $e=P/(k-1)\rho$; H – энтальпия, $H=E+P/\rho$; $\varphi_{1,s}$ и $\varphi_{2,s}$ – обратные времена динамической и тепловой релаксации, соответственно; N – количество фракций, на которые разбита вся совокупность частиц. Индекс s относится к частицам.

Система уравнений (1) замыкается уравнением состояния $P=\rho RT$.

Начальные условия

Для газа

$$\rho=\rho_н;\quad U=0;\ V=0;\ P=p_н.$$

Для частиц

$$\rho_s=0;\quad U_s=0;\ V_s=0;\ T_s=0.$$

Граничные условия для газа

Поверхность горения $y = L_2$:

$$\rho V=(1-z)m_T P^{\nu};\ U=0;\ \frac{k}{k-1}\frac{P}{\rho}+\frac{V^2}{2}=H_T.$$

Плоскость симметрии $y = 0$:

$$\frac{\partial P}{\partial y}=\frac{\partial \rho}{\partial y}=\frac{\partial U}{\partial y}=V=0.$$

Торцевая поверхность $x = 0$:

$$U=0.$$

Поверхность истечения $x = L_1$:

$$P=P_H.$$

Граничные условия для частиц

Поверхность горения $y = L_2$:

$$\rho_s = \frac{z}{1-z}\rho f_s;\ U_s = U;\ V_s = V;\ T_s = T\,.$$

Плоскость симметрии $y = 0$:

$$\left.\frac{\partial T_s}{\partial y}\right|_{y=+0} = \left.\frac{\partial T_s}{\partial y}\right|_{y=-0};\ \left.\frac{\partial \rho_s}{\partial y}\right|_{y=+0} = \left.\frac{\partial \rho_s}{\partial y}\right|_{y=-0};\ \left.\frac{\partial U_s}{\partial y}\right|_{y=+0} = \left.\frac{\partial U_s}{\partial y}\right|_{y=-0};\ V_s = 0\,.$$

Торцевая поверхность $x = 0$:

$$U_s = 0\,.$$

Поверхность истечения $x = L_1$: условия не задаются.

Здесь m_T, ν – константы закона скорости горения топлива (боковой поверхности стенки канала); z – массовая доля частиц; H_T – энтальпия топлива; f_s – функция распределения частиц по размерам.

Разностная схема

Для решения системы уравнений (1) использовался метод конечных объемов. Для этого исходная область решения разбивалась прямоугольной разностной сеткой на ячейки, для каждой из которых система уравнений (1) после интегрирования принимала вид

$$\frac{\mathbf{U}_{i,j}^{n+1} - \mathbf{U}_{i,j}^{n}}{\tau} + \frac{\mathbf{F}_{i+1/2,j} - \mathbf{F}_{i-1/2,j}}{h_x} + \frac{\mathbf{G}_{i,j+1/2} - \mathbf{G}_{i,j-1/2}}{h_y} = \mathbf{E}^{n+1}. \qquad (2)$$

Здесь потоки $\boldsymbol{F}$ и $\boldsymbol{G}$ для газа на гранях ячеек определялись по методу Ван-Лира [2], а для частиц – по методу А.Н. Крайко [3]. Целые индексы i и j относятся к центру ячейки, а полуцелые – к граням ячеек.

Если обозначить в системе разностных уравнений (2) конвективные комплексы

$$\frac{\mathbf{F}_{i+1/2,j} - \mathbf{F}_{i-1/2,j}}{h_x} + \frac{\mathbf{G}_{i,j+1/2} - \mathbf{G}_{i,j-1/2}}{h_y}$$

для уравнений импульса и энергии, соответственно, через A_i, B_i, C_i для газа и через A_{si}, B_{si}, C_{ss} для частиц, то система (2) может быть разрешена относительно параметров на n+1 временном слое следующим образом [4]:

$$U_i^{n+1} = \frac{A + \sum_{s=1}^{N} A_{si}\dfrac{\varphi_{1s}\tau}{1+\varphi_{1s}\tau}}{\rho_i^{n+1} + \sum_{s=1}^{N} \rho_{si}^{n+1}\dfrac{\varphi_{1s}\tau}{1+\varphi_{1s}\tau}};\qquad V_i^{n+1} = \frac{B_i + \sum_{s=1}^{N} B_{si}\dfrac{\varphi_{1s}\tau}{1+\varphi_{1s}\tau}}{\rho_i^{n+1} + \sum_{s=1}^{N} \rho_{si}^{n+1}\dfrac{\varphi_{1s}\tau}{1+\varphi_{1s}\tau}};$$

$$U_{si}^{n+1} = \frac{A_{si}}{\rho_{si}^{n+1}(1+\varphi_{1s}\tau)} + \frac{\varphi_{1s}\tau}{1+\varphi_{1s}\tau}U_i^{n+1};\ V_{si}^{n+1} = \frac{B_{si}}{\rho_{si}^{n+1}(1+\varphi_{1s}\tau)} + \frac{\varphi_{1s}\tau}{1+\varphi_{1s}\tau}V_i^{n+1}; \qquad (3)$$

$$T_i^{n+1} = \frac{C_i' + \sum_{s=1}^{N} C_{si} c_B \frac{\varphi_{2s}\tau}{1+\varphi_{2s}\tau}}{\rho_i^{n+1} c_V + \sum_{s=1}^{N} \rho_{si}^{n+1} c_B \frac{\varphi_{2s}\tau}{1+\varphi_{2s}\tau}}; \qquad T_{si}^{n+1} = \frac{C_{si}}{\rho_{si}^{n+1}(1+\varphi_{2s}\tau)} + \frac{\varphi_{2s}\tau}{1+\varphi_{2s}\tau} T_i^{n+1},$$

где $C_i' = C_i - \rho_i^{n+1} \frac{U_i^{n+12} + V_i^{n+12}}{2} - \tau \rho_{si}^{n+1} \varphi_1 \vec{V}_{si}^{n+1} (\vec{V}_i^{n+1} - \vec{V}_{si}^{n+1})$.

Данная схема является устойчивой для любых размеров частиц (обратных времен релаксации φ_s).

Невязка решения на каждом шаге по времени определяется по правилу:

$$d = \frac{1}{N \times M} \sum_{i=1}^{N} \sum_{j=1}^{M} \left(\left| \rho_{ij}^{n+1} - \rho_{ij}^{n} \right| + \left| U_{ij}^{n+1} - U_{ij}^{n} \right| + \left| V_{ij}^{n+1} - V_{ij}^{n} \right| + \left| P_{ij}^{n+1} - P_{ij}^{n} \right| \right).$$

Параллельный алгоритм решения задачи

Для распараллеливания полученного последовательного алгоритма (2)–(3) используется геометрическая декомпозиция, т.е. каждый процесс, получая исходные данные, находит решение поставленной задачи в отведенной ему геометрической подобласти. Рассматриваемая область решения проецируется на процессы, организованные в соответствии с декартовой топологией.

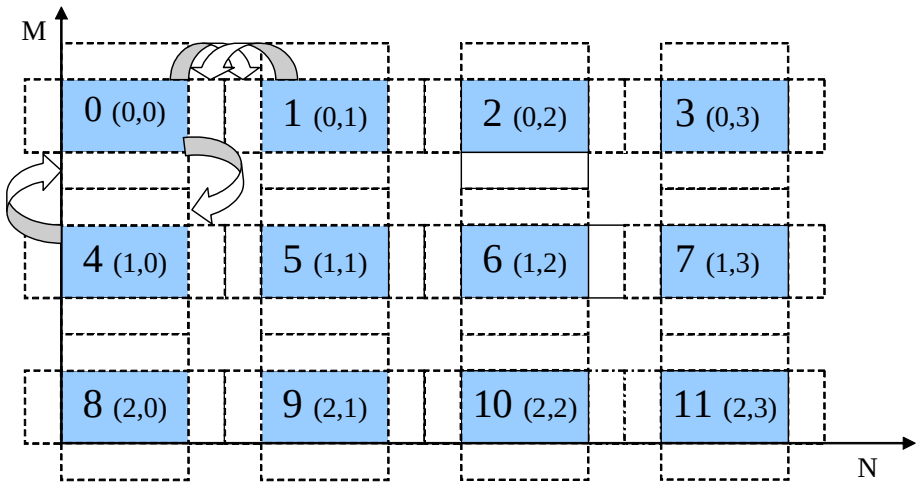


Рис. 2. Схема обмена данными между процессами в случае декартовой топологии

Для организации обмена информацией между процессами на каждом процессе вводятся четыре дополнительных граничных массива, в которые принимаются данные, посланные от соседних процессов. Эти данные необходимы для нахождения значений потоков на границах подобластей процессов. Обмен данными между процессами происходит по схеме, показанной на рис. 2.

Организация процессов в матрицу в случае декартовой топологии

Переменной SIZE присваиваем значение, равное количеству задействованных процессов, объединенных одним общим коммутатором icom, совпадающим с MPI_COMM_WORLD:

```
CALL MPI_COMM_SIZE(icom, SIZE, ierror)
```

Определяем номер текущего процесса RANK среди процессов, объединенных общим коммутатором icom:

```
CALL MPI_COMM_RANK(icom, RANK, ierror)
```

Задаем число процессов вдоль продольного направления:

```
ndims(1)=NX
```

Задаем число процессов вдоль поперечного направления:

```
ndims(2)=NY
```

Условие непериодичности границ области:

```
periods(1)=.false.,    periods(2)=.false.
```

Разрешаем производить перенумерацию процессов:

```
reorder=.true.
```

Устанавливаем новый коммутатор icom_2d, объединяющий процессы в матрицу:

```
CALL MPI_CART_CREATE(icom,2,ndims,periods,reorder,
                     icom_2d,ierr)
```

Получаем координаты id_coord текущего процесса в только что организованной матрице

```
CALL MPI_CART_COORDS(icom_2d,RANK,2,id_coord,ierr)
imap=id_coord(1)
jmap=id_coord(2)
```

Определяем номер процесса ill, находящегося слева от текущего процесса в координатах декартовой сетки

```
IF (imap.EQ.0) THEN
 ill=MPI_PROC_NULL
ELSE
 id_left(1)=imap-1
 id_left(2)=jmap
 CALL MPI_CART_RANK(icom_2d,id_left,ill,ierr)
ENDIF
```

Определяем номер процесса irr, находящегося справа от текущего процесса в координатах декартовой сетки

```
IF(imap.EQ.NX-1) THEN
 irr=MPI_PROC_NULL
ELSE
```

```
 id_right(1)=imap+1
 id_right(2)=jmap
 CALL MPI_CART_RANK(icom_2d,id_right,irr,ierr)
ENDIF
```

Определяем номер процесса ***ibb***, находящегося снизу от текущего процесса в координатах декартовой сетки

```
IF (jmap.EQ.0) THEN
 ibb=MPI_PROC_NULL
ELSE
 id_bottom(1)=imap
 id_bottom(2)=jmap-1
 CALL MPI_CART_RANK(icom_2d,id_bottom,ibb,ierr)
ENDIF
```

Определяем номер процесса ***itt***, находящегося снизу от текущего процесса в координатах декартовой сетки

```
IF(jmap.EQ.NY-1) THEN
 itt=MPI_PROC_NULL
ELSE
 id_top(1)=imap
 id_top(2)=jmap+1
 CALL MPI_CART_RANK(icom_2d,id_top,itt,ierr)
ENDIF
```

Обмен данными между процессами на одном шаге по времени

1) Заполняем буферный массив `S_right`, который будет послан «вправо» на процессор с номером `irr`.

2) Заполняем буферный массив `S_left`, который будет послан «влево» на процессор с номером `ill`.

3) Посылаем буферный массив `S_left` на процесс `ill`, который там будет записан в массив `R_left`:

```
CALL MPI_SENDRECV(S_left,Nbuf,ir8,ill,1,R_left,
        Nbuf,ir8,ill,1,icom,IREG(1),ierr1).
```

4) Посылаем буферный массив `S_right` на процесс `irr`, который там будет записан в массив `R_right`:

```
CALL MPI_SENDRECV(S_right,Nbuf,ir8,irr,1, R_right,
      Nbuf,ir8,irr,1,icom,IREG(2),ierr2).
```

5) Заполняем буферный массив `S_top`, который будет послан «вверх» на процессор с номером `itt`.

6) Заполняем буферный массив S_bottom, который будет послан «вниз» на процессор с номером ibb.

7) Посылаем буферный массив S_top на процесс itt, который там будет записан в массив R_top:

```
CALL MPI_SENDRECV(S_top,Nbuf,ir8,itt,2,R_top,
     Nbuf,ir8,itt,2,icom,IREG(3),ierr3).
```

8) Посылаем буферный массив S_bottom на процесс ibb, который там будет записан в массив R_bottom:

```
CALL MPI_SENDRECV(S_bottom,Nbuf,ir8,ibb,2,R_bottom,
     Nbuf,ir8,ibb,2,icom,IREG(4),ierr4).
```

9) Рассчитываем параметры на гранях ячеек.

10) Определяем минимальное значение шага по времени на каждом процессе, находим минимальный шаг по времени среди шагов, полученных от всех процессов, и рассылаем его на все процессы

```
CALL MPI_ALLREDUCE(TA_MIN,TA,1,ir8,MPI_MIN,
                   icom,ierror).
```

11) Рассчитываем параметры в центрах ячеек.

12) Определяем невязку на каждом процессе, находим сумму невязок со всех процессов, делим ее на число процессов size и рассылаем на все процессы

```
CALL MPI_ALLREDUCE(DD1,DD,1,ir8,MPI_SUM,icom,ierror)
DD=DD/SIZE.
```

Обсуждение результатов

Полученный параллельный алгоритм решения системы уравнений (1) был протестирован на вычислительном кластере ТГУ, состоящем из девяти двухпроцессорных узлов.

Расчеты проводились на разностной сетке размером 144×144, количество фракций принималось равным 10 и 50. Для вычисления ускорения полученного алгоритма определялось время выполнения 100 шагов по временной координате. При этом проводилось сравнение матричной и линейной декартовых топологий следующих размерностей: 2×2, 4×4 и 1×4, 1×16. В табл. 1 показано время работы программы в секундах и увеличение ускорения алгоритма, при работе программы на 8 узлах. Видно, что увеличение числа процессов в четыре раза приводит к увеличению ускорения, близкому трем в случае линейной организации процессов, и равному 3.5 – в случае матричной организации процессов. При этом скорость выполнения программы для матричной организации процессов оказалась выше, чем для линейной. Так, на 4 процессах это превышение составило 6% , а на 16 процессах – 33 %. Из результатов расчета также следует, что увеличе-

ние времени решения из-за увеличения числа фракций слабо зависит от способа организации процессов и определяется, в основном, их количеством. Пятикратное увеличение числа процессов привело к увеличению времени расчета в 5,1 раза на 4 процессах и в 5,5 раза на 16 процессах.

Таблица 1

Время выполнения программы на 8 узлах

10 фракций			50 фракций		
Число процессов	Линейка	Матрица	Число процессов	Линейка	Матрица
4	38,05	36,52	4	195,48	183,76
16	12,89	9,74	16	71,57	53,52
Увеличение ускорения	2,95	3,75	Увеличение ускорения	2,73	3,43

Поскольку загрузка узлов кластера неравномерная, ввиду коллективного его использования, то затруднительно получить адекватное время решения задачи на одном процессе. Так, на момент тестирования алгоритма с одним процессом для 50 фракций время работы программы на каждом из узлов (в секундах) было следующим: 788; 478; 268; 257; 241; 335; 241; 241.

Таблица 2

Время выполнения программы на 1 узле

10 фракций		
Число процессов	Линейка	Матрица
4	30,91	30,84
16	28,83	23,7

Для избегания влияния неравномерности загрузки узлов на результаты сравнения времени работы программы с линейной и матричной организации процессов были выполнены расчеты на одном узле. В табл. 2 приведено время решения задачи, полученное на одном из узлов с помощью 4 и 16 процессов. Видно, что способ организации процессов практически не влияет на время решения для 4 процессов, в то время как четырехкратное увеличение числа процессов ведет к экономии времени примерно на 20% в случае матричной организации.

Литература

1. Нигматулин Р.И. Основы механики гетерогенных сред. – М.: Наука, 1978. – 336 с.
2. Van Leer B. Towards the Ultimate Conservative Finite Difference Scheme III. Upstream centered Schemes for Hyperbolic Conservation Laws // J. Comp. Phys. – 1977. – Vol. 7, № 23. – P. 263–275.
3. Крайко А.Н. О поверхностях разрыва в среде, лишенной «собственного» давления // ПММ. – 1979. – Т. 43. – С. 500–511.
4. Миньков Л.Л., Шрагер Э.Р. Влияние доли несгоревших зерен дополнительного изделия на начальный участок работы ЭУ // Вычислительная газодинамика и горение твердых топлив. – Томск: Изд-во ТГПУ, 2001. – С. 160–166.
5. Корнеев В.Д. Параллельное программирование в MPI. – Новосибирск: Изд-во СО РАН, 2000. – 213 с.

ПАРАЛЛЕЛЬНАЯ РЕАЛИЗАЦИЯ МЕТОДА ЧАСТИЦ ДЛЯ РЕШЕНИЯ ЗАДАЧ ГИДРОДИНАМИКИ СО СВОБОДНЫМИ ГРАНИЦАМИ

В.В. Малышенко

Кемеровский государственный университет

Развитие компьютерной техники и технологий, ориентированное на создание суперкомпьютеров, кластерных установок, увеличение производительности, объема оперативной памяти и размера дисковых массивов, позволяет использовать ресурсоемкие численные методы для решения задач в полной нелинейной постановке. К данному классу методов относятся разновидности методов частиц: Smoothed Particle Hydrodynamics (SPH) method, Mixed Fixed Element Method (MFEM), MeshFree Method, метод свободно-лагранжевых частиц [1]. «Методами частиц» называются методы, использующие представление жидкости в виде набора материальных точек и решающие гидродинамические задачи, сопровождающиеся сильными нелинейными эффектами, значительными деформациями свободной границы, например, взаимодействие солитона с подводным уступом, накат уединенной волны на пологий берег, взаимодействие солитона с частично погруженным в жидкость телом. Использование частиц, не связанных сеткой, позволяет численно моделировать задачи гидродинамики со свободными границами, в которых происходит изменение связности области во время расчета, например в результате обрушения уединенной волны на береговую зону.

Методы частиц относятся к классу ресурсоемких методов, так как область течения жидкости представляется в виде набора материальных точек. Для каждой точки помимо пространственных координат необходимо хранить и другие характеристики: скорость, плотность, давление и т.д. Моделирование, например, нестационарной задачи о движении двух солитонов требует задания протяженного бассейна для изучения эффектов взаимодействия. В таких задачах количество частиц не менее 30–50 тысяч, это требует несколько гигабайт оперативной памяти. Время расчета одного шага по времени требует 1–4 мин. процессорного времени (Pentium IV 2,4 GHz). Поэтому необходимо использовать параллельные реализации методов частиц на суперкомпьютерах и кластерных установках.

Распараллеливание метода свободно-лагранжевых частиц [1] позволит эффективно и быстро решать задачи гидродинамики со свободными границами. В статье рассматривается решение нестационарной задачи о схло-

пывании выемки на свободной поверхности. Нестационарные задачи о схлопывании выемки на свободной поверхности всегда были объектом пристального внимания исследователей в силу того, что в процессе схлопывания образуется кумулятивная струя жидкости, обладающая высокой скоростью и сложной геометрией, вследствие чего данное явление становится трудным для математического моделирования. При решении задачи методами граничных элементов (МГЭ) и методами конечных элементов (МКЭ) невозможно производить исследование при отрыве от кумулятивной струйки капель жидкости и при обрушении волн, образующихся у основания выплеска, т.к. происходит изменение связности области расчета [2].

Постановка задачи

Рассмотрим движение идеальной несжимаемой жидкости в области V (рис. 1), ограниченной свободной поверхностью C_1 с полукруговой выемкой радиуса R, твердыми стенками и дном C_2, расположенным на глубине H.

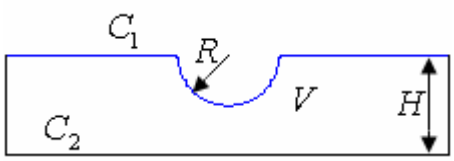


Рис. 1. Схема начальной геометрии области

В области V решается уравнение Эйлера:

$$\frac{d\vec{u}}{dt} = \frac{\vec{F}}{r},\ r \in V. \qquad (1)$$

Поле перемещений $\frac{d\vec{r}}{dt}$ определяется следующим образом:

$$\frac{d\vec{r}}{dt} = \vec{u},\ r \in V. \qquad (2)$$

Условие несжимаемости жидкости и условие непротекания на твердых стенках выписывается в виде

$$div\vec{u} = 0,\ r \in V. \qquad (3)$$

$$u_n = 0,\ r \in C_2, \qquad (4)$$

где n – внешняя нормаль к границе жидкости. Так как задача нестационарная, то требуется задание начального положения частиц жидкости и поля скоростей во всей области течения. Начальное поле скоростей $\vec{u}_0 = (u_0, v_0)$:

$$\begin{cases} u_0 = 0, \\ v_0 = 0. \end{cases}$$

От дифференциальной постановки задачи перейдем к вариационной постановке. Согласно принципу наименьшего принуждения Гаусса в каждый момент времени t жидкость будет двигаться так, чтобы минимизировать функционал:

$$S = \frac{1}{2}\int_V r(\dot{u} - \frac{F}{r})^2 dr \to \min, \tag{5}$$

$$div u = 0, \tag{6}$$

$$u_n|_{C_2} = 0. \tag{7}$$

Аппроксимируем объем V конечной системой материальных частиц с координатами r_k, массами m_k и скоростями u_k, $k = 1,K,N$. Обозначим шаг по времени t, $r_k^n = r_k(nt)$, $u_k^n = u_k(nt)$. Непрерывную задачу (5)–(7) аппроксимируем конечномерной задачей, которая имеет вид

$$S^n = \frac{1}{2}\sum_k m_k(\frac{\varphi_k - u_k^n}{t} - g)^2 \to \min,$$

$$div\varphi = 0, \tag{8}$$

$$\varphi_n|_{C_2} = 0,$$

где $\varphi_k = \varphi(r_k^{n+1})$, φ принадлежит некоторому конечномерному пространству H гладких соленоидальных функций. Для построения полностью консервативной схемы второго порядка аппроксимации по t, вводим предиктор

$$r_k^{n+1/2} = r_k^n + \frac{t}{2}u_k^n. \tag{9}$$

Тогда на шаге $n + \frac{1}{2}$ функционал будет иметь вид

$$S^{n+1/2} = \sum_k m_k(\frac{\varphi(r_k^{n+1/2}) - u_k^n}{t/2} - g)^2 \to \min, \tag{10}$$

а скорость и координаты точки на $n + 1$ шаге находятся по соответствующим формулам:

$$u_k^{n+1} = 2\varphi(r_k^{n+1/2}) - u_k^n,$$

$$r_k^{n+1} = r_k^n + t\varphi(r_r^{n+1/2}). \tag{11}$$

Разложим φ по базису пространства H

$$\varphi(r) = \sum_j b_j v_j(r). \tag{12}$$

Тогда, варьируя (10), получаем:

$$\sum_k m_k(\varphi_k - u_k^n - \frac{t}{2}g)v_i(r_k^{n+1/2}) = 0, i = 1K\,m, \tag{13}$$

где $\varphi_k = \varphi(r_k^{n+1/2})$.

Из (12), (13) получается система линейных уравнений относительно коэффициентов разложения b_j:

$$\sum_j a_{ij} b_j = \sum_k m_k (u_k^n + \frac{t}{2} g) v_i(r_k^{n+1/2}), \qquad (14)$$

где $a_{ij} = \sum_k m_k v_i(r_k^{n+1/2}) v_j(r_k^{n+1/2})$. Матрица a_{ij} является матрицей Грамма. Ее невырожденность при достаточно большом числе частиц обеспечивается линейной независимостью функций v_i.

Таким образом, один шаг по времени выглядит следующим образом. После предиктора (9) вычисляем матрицу и правую часть системы (14), решаем ее, из (10) находим гладкое поле θ, и, наконец, находим новые координаты и скорости частиц.

Построение базисных функций на основе кубического B-сплайна

Пусть на отрезке $[a,b]$ задано разбиение $\Delta x : a = x_0 < x_1 < \ldots < x_{m_x} = b$. Для построения кубического B-сплайна дополним сетку Δx узлами $x_{-3} < x_{-2} < x_{-1} < a$, $b < x_{m_x+1} < x_{m_x+2} < x_{m_x+3}$.

B-сплайн произвольной размерности задается рекуррентным соотношением, которое имеет следующий вид [6]:

$$B_i^0 = \begin{cases} 1, x \in [x_{i-1}, x_i], \\ 0, x \notin [x_{i-1}, x_i], \end{cases}$$

$$B_n^i(x) = \frac{x - x_i}{x_{i+n} - x_i} B_{n-1}^i(x) + \frac{x_{i+n+1} - x}{x_{i+n+1} - x_{i+1}} B_{n-1}^{i+1}(x).$$

Построим кубический B-сплайн, отличный от нуля, на интервале $(x_{i-2}; x_{i+2})$. В приложениях B-сплайны нечетных степеней удобно нумеровать по среднему узлу их интервалов-носителей. Искомый B-сплайн будем обозначать через $B_i(x)$ вместо $B_3^{i-2}(x)$. При решении задач гидродинамики в пространстве R^2 необходимо построение кубического B-сплайна по координате y, которое задается аналогичным образом.

Теперь построим базисные функции на основе B-сплайнов. В методе частиц базисные функции для решения задач гидродинамики со свободными границами имеют следующий вид:

$$v_{ij}(x,y) = rot(0,0,B_i(x)B_j(y)) = (\frac{dB_j(y)}{dy} B_i(x), -\frac{dB_i(x)}{dx} B_j(y), 0).$$

Базисная функция $v_{ij}(x,y)$ на основе B-сплайнов имеет ненулевое значение только в области $(x_{i-2}; x_{i+2}) \times (y_{j-2}; y_{j+2})$. Пусть эта область называется *основанием*.

Базисные функции такого вида удовлетворяют условию несжимаемости жидкости для функции скорости частиц.

Формирование матрицы Грамма

Эта матрица имеет некоторые особенности. Во-первых, при вычислениях она получается блочная, то есть только некоторые блоки матрицы имеют ненулевые элементы. Для удобства указания свойств матрицы Грамма в индексах базисной функции $v_{ij}(x,y)$ перейдем от двойного индекса ij к одинарному $-$ i, взаимнооднозначным соответствием $f : i * m_y + j \to i$.

Матрица $\{a_{ij}\}$, задаваемая по формуле $a_{ij} = \sum_k m_k v_i(r_k^{n+1/2}) v_j(r_k^{n+1/2})$, является матрицей Грамма. Если основания базисных функций v_i и v_j не пересекаются, то элемент a_{ij} будет заведомо нулевым, он так же будет нулевым, если в области пересечения оснований этих функций нет частиц. Тогда, если предопределять эти моменты, не будет необходимости в прямых вычислениях, что уменьшит количество операций, приходящихся на одну итерацию. Так же матрица $\{a_{ij}\}$ будет симметричной, так как элементы a_{ij} и a_{ji} будут вычисляться по тем же базисным функциям, причем положение частиц при этом не изменится.

Распараллеливание алгоритма метода частиц

Представление жидкости в виде набора частиц сопряжено с рядом проблем:

– количество частиц, аппроксимирующих объем жидкости в реальных расчетах составляет не менее 20–40 тыс. частиц;

– количество базисных функций – не менее 1000.

Эти ограничения накладывают определенные условия на используемые вычислители: не менее 512 Мбайт оперативной памяти и не менее 1 ГГц – тактовая частота процессора, но при таких условиях расчет задачи для данного, наперед заданного набора параметров происходит довольно долго – порядка 10 – 20 ч.

Поэтому целесообразно распараллелить данный численный алгоритм и в дальнейшем производить расчеты на суперкомпьютерах и кластерах. В качестве языка программирования выбран Фортран, параллельный интерфейс: MPI (Message Passing Interface) – интерфейс передачи сообщений, вычислительный ресурс – многопроцессорный гетерогенный вычислительный кластер, созданный на базе учебных компьютерных классов, производительностью 5 GFlops.

Анализ численного алгоритма показывает, что наиболее затратными по времени и ресурсоемкими по оперативной памяти являются следующие блоки: формирование матрицы и решение СЛАУ. На основе подходов к распараллеливанию, описанных в работе [3], данные блоки, занимающие

99% времени расчета, было принято решение распараллелить, а остальные изменить с учетом специфики параллельного программирования и необходимости синхронизации данных. Для решения СЛАУ используются разработанные и описанные в работе [4] параллельные алгоритмы численных методов решения СЛАУ.

Распараллеливание дополнительного блока вычисления дополнительной матрицы

Матрица $Mv\left(i,r_k\right)$, содержащая значения базисных функций в точках $v_i\left(r_k^{n+1/2}\right)$, разбивается на np равных частей по числу процессоров, на которых запускается параллельное приложение.

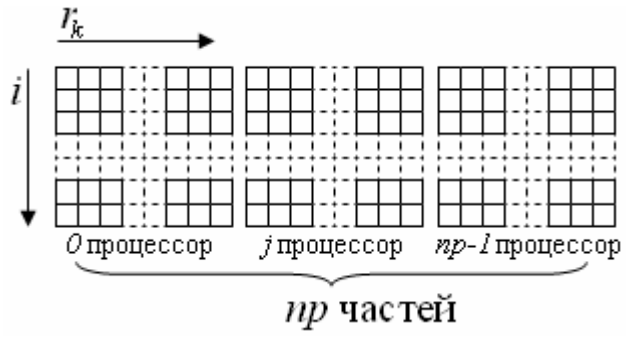


Рис. 2. Блочное разбиение матрицы $Mv\left(i,r_k\right)$ по процессорам

Нулевой процессор вычисляет значение 16 базисных функций $i = 1,\text{K},16$ в точках $r_k, k = 1,\text{K},\frac{N}{np}$, второй – в точках $r_k, k = \frac{N}{np} + 1,\text{K},2?\frac{N}{np}$ и т.д., последний – в точках $r_k, k = \frac{N}{np}?(np - 1) + 1,\text{K},\frac{N}{np}$. После вычисления производится пересылка полученных данных между процессорами. Данное разбиение матрицы $Mv(i,r_k)$ позволяет увеличить скорость вычисления почти в np раз.

Распараллеливание блока формирования матрицы A и вектора правой b части производилось с учетом того, что система линейных алгебраических уравнений $Ab = f$ решается параллельным методом Гаусса с выбором ведущего элемента и того, что язык программирования Фортран хранит матрицы по столбцам. Поэтому процессор p_i формирует свой блок столбцов $a_{ij}, i = 1 + id?\frac{m}{np}, (id + 1)?\frac{m}{np}, j = 1,m$. Это позволяет, не производя пересылок между процессорами, перейти к решению СЛАУ методом Гаусса с выбором ведущего элемента.

Тестирование численного алгоритма

Для тестирования метода была выбрана задача Овсянникова о деформации жидкого эллипса [5]. Данная задача принадлежит к классу задач со свободными границами. Задача формулируется следующим образом: в начальный момент времени область расчета представляет собой круг радиуса 1, в который заключена несжимаемая жидкость. Деформация круга в эллипс начинается под действием начального распределения скоростей в отсутствие внешних сил. Для обеспечения несжимаемости необходимо, чтобы площадь эллипса оставалась постоянной, т.е. $ab = 1$ во все время расчета, где a, b – полуоси эллипса.

Начальное поле скоростей следующее:

$$\begin{cases} u_0 = a'(0)/a(0)\,x, \\ v_0 = -a'(0)/a(0)\,y. \end{cases}$$

С ростом t круг деформируется в эллипс $\frac{x^2}{a^2(t)} + a^2(t)y^2 = 1$, большая полуось которого $a(t) \to \infty$, а малая $\frac{1}{a(t)} \to 0$ при $t \to \infty$.

Коэффициент $a(t)$ находится из решения задачи

$$\begin{cases} \left(a^4(t)+1\right)a(t)a''(t) = 2\left(a'(t)\right)^2, \\ a(0) = 1, \\ a'(0) = 1. \end{cases} \qquad (15)$$

Обыкновенное дифференциальное уравнение (15) второго порядка было решено методом Рунге–Кутта–Фельберга с точностью $O\left(t^4\right)$. Полученные значения $a(t)$ при $t \in [0;6,5]$ использовались для построения свободной границы жидкого эллипса и сравнения с численными расчетами.

Задача решалась только в верхней полуплоскости ввиду симметрии. Количество жидких частиц равно $N = 1700$, шаг сетки, в узлах которой были помещены частицы, $h = 0,05$, количество базисных функций – $m = 300$, шаг разбиения сеток Δx и Δy – $h = 0,2$. В расчете использовались базисные функции вида: $v_i(x,y) = rot(0,0,B_i(x)B_j(y)(y^3 + y^2 + y))$.

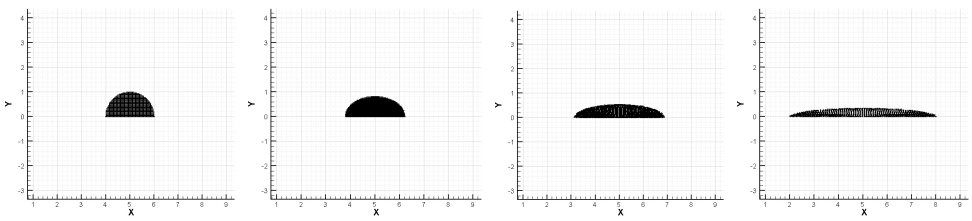


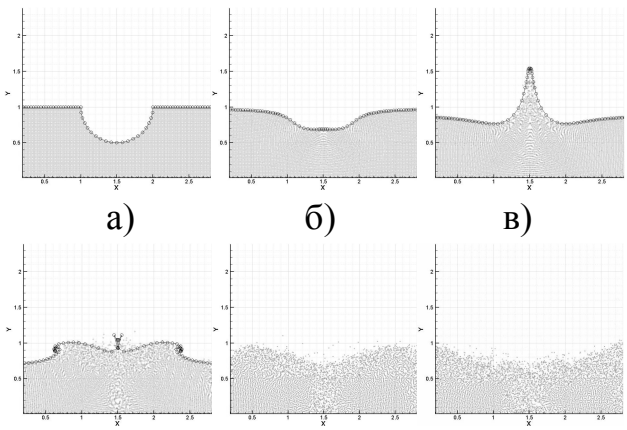
Рис. 3. Распределение жидких частиц в моменты времени $t = 0$ с, $t = 0,3$ с, $t = 0,9$ с и $t = 1,2$ с соответственно

Расчеты производились до времени $t = 6,5$ с, что составляет более 1000 итерационных шагов. Относительная погрешность отклонения $a(t)$ не превысила 1%. При этом абсолютная погрешность кинетической энергии равна 0,0024, а относительная – $5,41 * 10^{-8}$ процента. Выполненные расчеты показывают высокую точность полученного решения. На рис. 3 приведены результаты расчета задачи Овсянникова, при заданном начальном распределении скорости $\vec{v}_0 = (u_0, v_0) = (x, -y)$.

Решение задачи об эволюции полукруговой выемки

Количество жидких частиц равно $N = 6512$, шаг сетки, в узлах которой были помещены частицы, $h = 0,05$, количество базисных функций – $m = 60 ? 35 = 2100$, шаг разбиения сеток Dx и Dy – $h = 0,2$. В расчете использовались базисные функции, основанные на кубических B-сплайнах [6], вида: $v_{ij}(x,y) = rot(0,0,B_i(x)gB_j(y)g(y^3 + y^2 + y))$.

Для подтверждения достоверности результатов расчета, представленного на рис. 4, приведено сравнение полученных форм свободной поверхности с результатами, полученными при решении данной задачи методом граничных элементов [1], для соответствующих моментов времени. Глубина бассейна выбиралась равной $H = 2R$.



а) б) в)

г) д) е)

Рис. 4. Положение свободной границы и распределение жидких частиц в моменты времени $t = 0,0c$ $t = 0,5$ с, $t = 1,0$ с, $t = 2,0$ с, $t = 2,5$ с и $t = 3,0$ с соответственно

На рис. 4, а–г черным цветом показано положение свободной границы, полученной при решении поставленной задачи методом граничных элементов, серым цветом – положение частиц, полученных в результате решения задачи представленным методом. Сравнение показало количест-

венное и качественное совпадение результатов данной работы и работы [2]. Однако в связи с возникающими перехлестами свободной границы и образованием многосвязности расчетной области провести расчет МГЭ далее не удается. Метод частиц не критичен к указанным проблемам и позволяет провести расчет далее (после возникающих обрушений) – рис. 4, д, е.

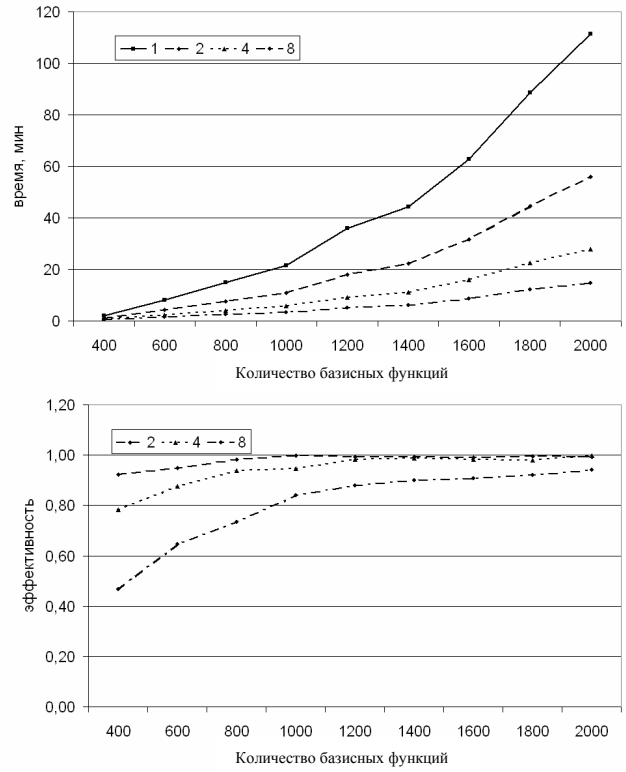


Рис. 5. Время и ускорение расчета параллельного алгоритма в зависимости от количества базисных функций

Решение задач гидродинамики со свободными границами параллельным методом частиц позволяет сократить время вычисления в среднем в 9 раз на 10 процессорах (рис. 5). Высокая степень параллелизма численного алгоритма позволяет эффективно проводить расчеты как на кластерах с низкой латентностью сетью, так и на кластере с сетевой инфраструктурой FastEthernet. Созданный параллельный программный комплекс в дальнейшем будет использоваться при решении задач взаимодействия поверхностных волн жидкости с препятствиями.

Литература

1. Франк А.М. Дискретные модели несжимаемой жидкости. – М.:Физматлит, 2001. – 181 с.
2. Afanas’ev K.E., Korotkov G.G. The development of a semi-circular cavity on the free surface in both plane and axisymmetrical cases // The Interna-

tional Summer Scientific School «High Speed Hydrodynamics», June 2002, Cheboksary. – P. 219–225.

3. Афанасьев К.Е., Стуколов С.В. КМГЭ для решения плоских задач гидродинамики и его реализация на параллельных компьютерах: Учебное пособие. – Кемерово: КемГУ, 2001. – 208 с.
4. Афанасьев К.Е., Домрачев В.Г., Ретинская И.В. и др. Многопроцессорные системы: построение, развитие, обучение / Под ред. А.Н. Тихонова. – М.: КУДИЦ-ОБРАЗ, 2005. – 224 с.
5. Овсянников Л.В. Общие уравнения и примеры: Задачи о неустановившемся движении жидкости со свободной границей. – Новосибирск: Наука, 1967. – С. 5–75.
6. Завьялов Ю.С., Квасов Б.И., Мирошниченко В.Л. Методы сплайн-функций. – М.: Наука, 1980. – 352 с.

О МЕТОДАХ РАЗРАБОТКИ ПАРАЛЛЕЛЬНЫХ КОМБИНАТОРНЫХ АЛГОРИТМОВ

Н.Е. Тимошевская

Томский государственный университет

Введение

Под *комбинаторными задачами* здесь подразумеваются перечислительные и поисковые задачи на конечных *комбинаторных множествах*, элементами которых служат *комбинаторные объекты*, представляющие собой комбинации элементов других конечных (возможно, тоже комбинаторных) множеств – перестановки, размещения, разбиения, подмножества, и т.п. *Перечислительная задача* заключается в перечислении всех элементов некоторого подмножества конечного комбинаторного множества, а *поисковая* – в нахождении в последнем элемента с заданными свойствами. Алгоритмы решения комбинаторных задач называются *комбинаторными алгоритмами*.

То обстоятельство, что решение комбинаторной задачи подразумевает вычисления в пределах конечного множества, делает методы и алгоритмы непрерывной математики фактически не применимыми к ней. За редким исключением перебор и опробование элементов основного комбинаторного множества является единственным способом решения комбинаторной задачи. Различные методы построения комбинаторных алгоритмов нацелены на сокращение и ускорение перебора и отличаются друг от друга тем, как они это делают. Почти все комбинаторные алгоритмы имеют экспоненциальную сложность. Вместе с тем на практике необходимо уметь решать комбинаторные задачи достаточно больших размерностей.

Параллельные вычисления сегодня широко используются при решении различных задач, требующих больших вычислительных ресурсов [1, 2]. К таковым, несомненно, относятся и комбинаторные задачи. Один из способов уменьшения времени решения комбинаторной задачи состоит в том, чтобы разбить комбинаторное множество на классы и исследовать эти классы одновременно, т.е. параллельно, на нескольких вычислительных устройствах. В данной работе будут рассмотрены методы разработки параллельных алгоритмов трех классов – перечислительных, использующих дерево поиска и динамического программирования. Обзор составлен по результатам зарубежных публикаций и исследований автора. Среди алгоритмов, использующих дерево поиска, особое внимание уделяется способам организации параллельного обхода дерева в глубину, отмечаются особенности, возникающие при распараллеливании алгоритмов поиска опти-

мальных решений, и отдельно рассматриваются алгоритмы ветвей и границ.

1. Параллельные перечислительные алгоритмы

В алгоритмах перечисления комбинаторных объектов, как правило, объекты порождаются в некотором порядке путем преобразования текущего объекта в следующий за ним по определенным правилам [3, 4]. Для распараллеливания данной схемы множество перечисляемых объектов следует предварительно разбить на классы и перечислять объекты в различных классах независимо и параллельно. Однако когда речь идет о множестве комбинаторных объектов, в отличие, скажем, от числового множества, провести такое разбиение в явном виде не представляется возможным.

В работах [5, 6] для решения этой проблемы предлагается метод нумерации, суть которого заключается в сопоставлении перечисляемым комбинаторным объектам их номеров из натурального ряда в соответствии с установленным на множестве объектов линейным порядком. Последний, естественно, должен допускать восстановление объекта по его номеру, причем время, требуемое на восстановление, должно быть пренебрежимо малым по сравнению со временем, затрачиваемым на перечисление всех объектов одного класса.

Пусть, таким образом, на множестве перечисляемых комбинаторных объектов задан некоторый линейный порядок и имеются подходящие алгоритм восстановления объекта по его номеру и алгоритм перечисления в заданном порядке требуемого количества объектов, начиная с некоторого. Динамическую балансировку нагрузки процессоров можно обеспечить, разбив все множество объектов на большое число классов сравнительно малой мощности и выдавая полученные классы на обработку в ответ на запросы освободившихся процессоров. Этот подход может использоваться для организации распределенных вычислений в большой вычислительной сети.

Если же используемая вычислительная система не изменяется в динамике, то можно использовать статическую балансировку загрузки процессоров, разбив все множество объектов на классы, отвечающие по мощности производительности процессоров в системе, например, следующим образом. Пусть имеется система из s процессоров с числовыми характеристиками их быстродействия $w_1 \geq w_2 \geq \ldots \geq w_s$ соответственно. Если z − число всех перечисляемых объектов и $w = \sum_{j=1}^{s} w_j$, то число z_j объектов, перечисляемых j-м процессором, и номер I_j первого из них вычисляются как $z_j = \left[\frac{w_j}{w}\right] \cdot z$ для $j = 1, \ldots, s-1$, $z_s = z - \sum_{j=1}^{s-1} z_j$ и $I_j = \sum_{k=1}^{j-1} z_k + 1$ для всех $j = 1, \ldots, s$.

Возможны различные пути организации параллельной работы процессоров. Первый вариант следующий. Для каждого $j = 1, \ldots, s$ один из процессоров по заданным $w_1, w_2, \ldots, w_s, z$ вычисляет значения I_j, z_j и посылает их j-му процессору. Затем на j-м процессоре с помощью алгоритма восстановления строится объект с номером I_j и запускается алгоритм, перечисляющий z_j следующих за ним объектов. Во втором варианте каждому процессору передается уже восстановленный объект, и на процессоре работает только алгоритм перечисления. В третьем – значения I_j и z_j не передаются, но вычисляются j-м процессором по известным всем числам $w_1, w_2, \ldots, w_s, z$.

В работе [6] представлены параллельные алгоритмы перечисления перестановок и сочетаний методом нумерации. Установленная в эксперименте эффективность этих алгоритмов в среднем равна 0,94 и 0,87 соответственно. Алгоритм параллельного перечисления разбиений множества методом нумерации также имеет высокую эффективность (порядка 0,96).

Следует отметить, что для достижения высокой эффективности при использовании статической балансировки необходимо, чтобы в течение всего времени счета параллельной программы установленное соотношение производительностей процессоров сохранялось неизменным, что на практике не всегда имеет место (в частности, когда система обслуживает многих пользователей). Заметим также, что применение данного метода ограничено кругом задач, допускающих возможность нумерации требуемых в задаче объектов так, что по номеру можно легко восстановить сопоставленный ему объект.

2. Параллельные алгоритмы, использующие обход дерева поиска

Благодаря комбинационной структуре объектов, во многих комбинаторных задачах в рассмотрение вводится *дерево поиска*, которое используется в качестве модели пространства возможных решений, а его обход – как модель алгоритма поиска решения. Дерево поиска может иметь настолько большие размеры, что его невозможно представить в компьютере целиком. По этой причине дерево обрабатывают поэлементно. На однопроцессорном компьютере обход дерева выполняется последовательно – узлы дерева воспроизводятся один за другим по одному. В зависимости от порядка воспроизведения узлов различают обходы в глубину, в ширину и в порядке «наилучшего выбора».

2.1. Методы параллельного обхода дерева в глубину

При обходе в глубину в каждом поддереве дерева сначала проходится корень, а затем его поддеревья.

Идея распараллеливания процедуры обхода дерева, прослеживаемая почти во всех работах, состоит в том, что дерево поиска разбивается на

множества поддеревьев и их обход поручается разным процессорам. Проблема заключается в том, как разбить (статически или динамически) дерево так, чтобы обеспечить высокую эффективность распараллеливания, на снижение которой влияют многие факторы, в том числе неравномерная загрузка процессоров системы, накладные расходы на коммуникации, конкуренция процессоров за доступ к разделяемым данным, избыточность просматриваемой области поиска.

Статический подход к распределению поддеревьев по процессорам эффективен только для деревьев с предсказуемой структурой. В работе [8] этот подход используется для обхода в глубину дерева поиска разбиений. Поддеревья распределяются по процессорам в соответствии с их номерами. Экспериментально показано, что при правильном выборе яруса, содержащего корневые вершины распределяемых поддеревьев, удается достигнуть максимального ускорения.

В большинстве же комбинаторных задач дерево поиска носит нерегулярный, непредсказуемый характер, и для обеспечения хорошей эффективности распараллеливания его обхода требуется выполнение динамических процедур выделения и распределения поддеревьев между процессорами. Если какой-либо из процессов, а каждый процесс исполняется на отдельном процессоре, завершает обход назначенных ему поддеревьев раньше других, то он становится *свободным* и должен получить часть работы какого-либо занятого процесса, называемого в данном случае *источником*.

В [7] дается классификация методов обхода по следующим правилам распределения поддеревьев: 1) разбиение и передача инициируются источником [9, 10]; 2) разбиение инициируется источником, а передача выполняется по запросу свободного процесса [11, 12]; 3) разбиение и передача выполняются по запросу свободного процесса [13–17].

В первой группе методов процессы, выступающие источниками, генерируют и рассылают корни поддеревьев независимо от того, в каком состоянии находятся другие процессы. Если обход дерева сопровождается выполнением в вершине большого объема вычислений, то, наверное, оправданным будет подход, описанный в работе [10], где корни поддеревьев источник рассылает случайно выбранным процессам при каждом ветвлении, что в некоторой степени обеспечивает равномерную загрузку процессоров. Однако в такой схеме коммуникации сильно перегружают сеть и число вершин на каждом процессоре может расти неограниченно, что приводит к ограничениям по памяти.

В работе [9] представлен вариант этой схемы для вычислительных систем с архитектурой butterfly или гиперкуба, в котором порождаемые вершины бинарного дерева распределяются по соседним процессорам. Авторами метода показывается, что с высокой степенью вероятности время,

требуемое для параллельного обхода дерева с M вершинами на P процессорах, равно $O(M/P + \log P)$. Эта схема локализует все пересылки и поэтому оказывается более эффективной по сравнению с представленной в [10]. Введение ограничения, запрещающего передавать вершины, расположенные ниже заданного *граничного* яруса, позволяет повысить эффективность описанных методов [18]. Все вершины, расположенные ниже граничного яруса, проходятся породившим их процессом, что приводит к снижению объема коммуникаций.

Во второй группе методов корневые вершины поддеревьев (поддеревья) генерируются источниками независимо, но пересылаются свободным процессам, только по запросу от них. В работе [12] представлен метод *назначаемых поддеревьев*, в котором генерирование корневых вершин возложено на отдельно выделенный процесс, называемый *управляющим*, а все остальные процессы называются *рабочими*. Управляющий процесс вначале генерирует большое число корневых вершин поддеревьев, затем назначает каждому процессу по одному поддереву, после обхода которого рабочий процесс получает следующую корневую вершину и т.д. В работе [11] для генерирования корневых вершин используется несколько процессов-источников. Каждый рабочий процесс приписан к одному из них. Если у источника заканчиваются поддеревья, то его рабочие процессы приписываются к другому источнику. Такой подход позволяет избежать перегрузки управляющего процесса. В [19] предлагается метод *fixed-packet DFS*, в котором на первом этапе все процессы выполняют обход дерева в ширину до некоторого яруса. Множество вершин последнего яруса разбивается на классы по числу процессов. На втором этапе каждый процесс начинает обход в глубину поддеревьев, корневые вершины которых лежат в его классе. Когда процесс освобождается, он направляет кому-нибудь запрос. В ответ источник посылает освободившемуся процессу часть корневых вершин из своего класса. Этот метод успешно применяется в итеративном поиске.

К третьей (самой распространенной) группе относятся методы, в которых и разбиение, и передача поддеревьев выполняются по запросам от свободных процессов. Эти методы представлены в работах [12–14, 18, 20, 28, 29]. Опишем их общую схему. Каждый из процессов выполняет обход в глубину назначенных ему одного или нескольких поддеревьев, при этом периодически проверяет наличие запросов от других процессов. Когда процесс завершает обход назначенных ему поддеревьев, он становится свободным и выбирает для себя процесс, которому посылает запрос на работу. Процесс, получивший запрос, становится источником и либо посылает отказ, либо выделяет и посылает часть оставшихся у него поддеревьев свободному процессу.

Методы последней группы различаются способами выбора процесса-источника, выделения и начального распределения поддеревьев и условиями посылки отказа.

Описано несколько способов (стратегий) выбора процесса-источника, т.е. того, к кому направляется запрос свободного процесса. В стратегии *случайного голосования* (random polling) [14, 17, 21, 29] выбор осуществляется случайным образом. Если в ответ от выбранного процесса приходит отказ, то случайный выбор повторяется. Минусом этой стратегии является большое число отказов. В стратегии *глобальной кольцевой очереди* [22] имеется глобальный указатель на процесс, к которому надо направлять запрос. После обработки запроса указатель – переменная, содержащая номер процесса, переходит к следующему процессу (увеличивается по модулю числа процессов). В [22] показано, что использование глобального указателя позволяет значительно уменьшить число запросов для большого числа видов деревьев, однако при большом числе процессов возникает конкуренция за доступ к глобальному указателю. В стратегии *оптимального выбора* [12] запрос направляется к наиболее загруженному процессу.

При выполнении обхода в глубину каждым процессом используется стек, в который помещаются и затем по мере обхода извлекаются корневые вершины еще не пройденных поддеревьев. Таким образом, можно сказать, что в каждый момент времени процесс имеет набор поддеревьев, корневые вершины которых находятся в стеке. В ответ на запрос освободившегося процесса процесс-источник должен выделить часть своих поддеревьев, чтобы передать их свободному процессу. В стратегии *node splitting* выделяется одно поддерево – путем исключения одной вершины из стека [12, 28, 29]. В стратегии *stack splitting* выделяется несколько поддеревьев – путем расщепления стека на два [18, 20, 22, 29]. Согласно второй стратегии свободный процесс получает стек вершин. Стек процесса-источника расщепляется так, чтобы в стеке свободного процесса и в стеке источника оказалось примерно одинаковое число вершин и чтобы в каждом стеке были вершины разных ярусов. Преимущество расщепления стека авторы этого метода видят в том, что оно должно приводить к равномерному распределению вычислений между свободным процессом и источником.

Начальное распределение поддеревьев между процессами можно организовать по-разному. В некоторых методах изначально все дерево назначается одному процессу, а остальные являются свободными. Затем дерево постепенно динамически распределяется между процессами [14, 17, 18]. Этот подход характеризуется большим числом коммуникаций уже на начальном этапе. В методе *выделяемых поддеревьев*, описанном в [12], один процесс вначале генерирует столько корневых вершин, сколько есть процессов, выполняя обход дерева в ширину. Полученные в результате обхода в ширину корневые вершины располагаются максимально близко к корню.

Условия посылки отказа определяются скорее не методом параллельного исполнения поиска, а задачей и алгоритмом, к которому он применяется. Одно из самых простых условий накладывается на число уже полно-

стью пройденных ярусов в дереве процесса-источника, а именно: если предположительное число оставшихся ярусов очень мало (насколько мало зависит от размерностей задачи и характеристик используемой системы), то высылается отказ.

С помощью описанных выше методов разными авторами построены параллельные алгоритмы решения таких задач, как задача коммивояжера, тур коня, игра в пятнадцать, проверка тавтологии, автоматическое генерирование тестов, задача о рюкзаке и др. Алгоритмы исследовались на многопроцессорных системах различных типов, и почти для всех задач тем или иным методом достигалось линейное ускорение.

2.2. Параллельный поиск оптимального решения

В задачах поиска оптимального (для определенности будем говорить минимального) решения используется некоторая целевая функция – характеристика решения. На ее основе, как правило, строится функция нижней оценки стоимости частичного решения. В каждой вершине дерева поиска проводится анализ, связанный с полученным на данный момент лучшим решением, по результатам которого продолжается ветвление дерева из рассматриваемой вершины либо выполняется возврат на предыдущий уровень.

При поиске оптимального решения параллельным обходом дерева в глубину необходимо информировать все процессы о находимых решениях. Методы поиска отличаются друг от друга тем, как они это делают [7, 13]. В системах с общей памятью можно хранить глобальное лучшее решение среди разделяемых данных, к которым все процессы имеют доступ. В ряде методов, ориентированных на системы, не имеющие общей памяти, текущее решение хранится в локальной памяти каждого процесса. Когда какой-либо из процессов находит решение лучше хранимого у него, он обновляет его у себя и передает всем остальным процессам, которые, если надо, обновляют свои решения. Заметим, что если текущее решение, хранящееся у процесса, хуже глобального текущего решения, то это лишь снижает эффективность поиска, но не влияет на корректность найденного в итоге решения. Для ряда задач и архитектур известны алгоритмы с линейным ускорением. Например, в работе [23] дается изложение параллельных алгоритмов и результатов исследований для задачи оптимальной компоновки схем.

2.3. Аномалии ускорения в параллельных алгоритмах обхода дерева поиска

Если решается задача поиска и достаточно найти хотя бы одно решение, то во всех методах параллельный поиск прекращается, как только какой-нибудь из процессов находит решение. В задачах такого типа наиболее

заметны эффекты несовпадения просматриваемых областей поиска для параллельного и последовательного алгоритмов и, как следствие, выполнения ими работы разного объема. В параллельном случае он может оказаться как меньше, так и больше. *Фактор избыточности поиска* R определяется как отношение объема работы (числа просмотренных вершин) последовательного алгоритма к объему работы параллельного. Тогда верхняя граница ускорения для p процессов есть pR. В некоторых случаях фактор избыточности может оказаться меньше единицы, что приводит к суперлинейному ускорению. Если фактор избыточности поиска в среднем меньше единицы, это означает, что решения в дереве поиска распределены неравномерно или что выбранный последовательный алгоритм не является лучшим для этой задачи.

В работе [16] проведен глубокий анализ этой проблемы. Предлагаются абстрактные модели пространства поиска, просматриваемого алгоритмами параллельного поиска в глубину двух типов: простого и с упорядочиванием порождаемых вершин. Анализируется и сравнивается среднее число узлов, просматриваемых последовательным и параллельным алгоритмами. Для простого поиска доказывается, что если распределение решений по областям поиска разных процессоров одинаково, то области поиска последовательного и параллельного алгоритмов примерно одинаковы, и если распределения решений по областям поиска разных процессоров различаются, то область поиска последовательного алгоритма больше области поиска параллельного. Для упорядоченного поиска доказывается, что число узлов, просматриваемых параллельным алгоритмом, не больше числа узлов, просматриваемых последовательным.

В работах [24, 25] представлен метод, в котором ветвление в вершинах выполняется по приоритетам так, что параллельный алгоритм посещает узлы дерева в том же порядке, что и последовательный. Такой подход позволяет значительно повысить ускорение при использовании большого числа процессов при решении достаточно больших задач. Однако необходимость слежения за приоритетами делает эту схему слабо масштабируемой.

2.4. Итеративный поиск в глубину (IDA)*

Итеративный последовательный поиск в глубину чаще применяется для решения задач оптимизации и бывает в двух вариантах [26]. В первом – с ограничением по глубине – на каждой итерации задается номер яруса, до которого выполняется поиск, в надежде, что длина пути, представляющего решение, меньше этого яруса. Этот вариант применяется тогда, когда оптимальное решение задается кратчайшим путем в дереве. От итерации к итерации граничный ярус увеличивается. Во втором варианте на каждой итерации увеличивается значение границы, т.е. если оценка

стоимости решения в рассматриваемой вершине превышает заданную границу, то ветвления из нее не производится.

Параллельное исполнение итеративного поиска встречается в литературе в двух версиях. В первой – различные процессы независимо работают на разных итерациях. Однако при решении задач оптимизации быть уверенным в оптимальности решения, найденного на некоторой итерации, можно только после того, как завершится поиск на всех итерациях. Во второй – на каждой итерации выполняется параллельный поиск в глубину с указанными ограничениями. После завершения итерации, если решение еще не найдено, то специально выделенный процесс задает новое ограничение и запускается следующая итерация [19].

2.5. Методы параллельного выполнения алгоритмов ветвей и границ

В алгоритмах ветвей и границ на каждом шаге для ветвления в дереве поиска выбирается наиболее перспективная из еще не обработанных вершин. Одним из подходов к распараллеливанию этих алгоритмов является хранение глобального списка вершин и выдача из него самой перспективной вершины запросившему процессу. После выполнения ветвления процесс оценивает полученные вершины и те из них, которые удовлетворяют граничным условиям, помещает в глобальный список. Методы такого типа относятся к централизованным [7, 13, 21, 27]. Они отличаются друг от друга только используемыми для хранения списка структурами данных. Для систем, не имеющих общей памяти, такой подход требует огромных расходов на передачи вершин после каждого ветвления.

Альтернативный подход заключается в разбиении дерева на поддеревья, которые независимо исследуются разными процессами. Каждый из процессов хранит собственный список необработанных вершин в его поддереве. Этот подход не требует больших коммуникационных расходов, но может привести к избыточности просматриваемой области поиска и несбалансированности загрузки процессов [35].

Встречаются методы, использующие композицию этих двух подходов. С некоторой периодичностью процессы составляют глобальный список вершин из своих локальных списков и выбирают наиболее перспективные вершины в качестве корневых для дальнейшего исследования. Чем чаще они обмениваются данными, тем меньше потерь, связанных с балансировкой и избыточностью области поиска, но больше накладных расходов на коммуникации, и наоборот. В работе [30] показано, что для некоторых алгоритмов ветвей и границ существует такое k, что одновременное ветвление более чем в k вершинах не дает большого преимущества. Такие значения экспериментальным путем были установлены для задачи о рюкзаке.

При исполнении параллельных алгоритмов ветвей и границ также наблюдаются аномалии ускорения [30, 31]. Одной из причин их возникновения является то, что многие вершины в дереве могут иметь одинаковую оценку, хотя большинство из них не лежит на пути решения. Следовательно, если параллельный алгоритм раньше начнет исследовать часть дерева, содержащую правильный путь, то будет получено суперлинейное ускорение, в противном случае – наоборот.

3. Параллельные алгоритмы динамического программирования

Метод динамического программирования состоит в сведении исходной задачи к решению некоторых ее подзадач меньшего размера и использовании табличной техники для сохранения уже найденных ответов. Решение подзадач при этом происходит в порядке возрастания их размеров — от меньших к большим до тех пор, пока не будет решена исходная задача. Некоторый обзор методов распараллеливания алгоритмов динамического программирования можно найти в работе [7].

Пусть последовательный алгоритм разбивает задачу на подзадачи нескольких уровней так, что решение подзадач одного уровня зависит только от решений подзадач предыдущих уровней. Тогда в самом простом параллельном исполнении на каждом уровне задачи распределяются по процессорам, которые решают их и рассылают остальным процессорам полученные решения для использования в решении задач следующего уровня [21, 33]. В результате каждому процессу известны решения всех подзадач текущего уровня. Если уровень содержит N подзадач, то каждому из p процессоров назначается по $[N/p]$ подзадач. В этом методе не может быть задействовано больше процессоров, чем максимальное число подзадач одного уровня.

Использование нескольких процессоров для вычисления стоимости каждой подзадачи иногда оказывается более эффективным [32]. В другой технике распараллеливания решение подзадач на разных уровнях ведется по конвейерному принципу. Для решения некоторой подзадачи не требуются решения всех подзадач предыдущих уровней, вместо этого ожидается решение только тех подзадач, от которых непосредственно зависит решение рассматриваемой. Эта техника используется для решения задачи о расстановке скобок в произведении матриц [34]. Последовательный алгоритм решения этой задачи имеет сложность $O(n^3)$. При использовании $O(n^2)$ процессоров вычисления выполняются за время $O(n)$.

Литература

1. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – СПб.: БХВ-Петербург, 2002. – 608 с.

2. Гергель В.П., Стронгин Р.Г. Основы параллельных вычислений для многопроцессорных вычислительных машин. – Нижний Новгород: Изд-во ННГУ им. Н.И. Лобачевского, 2003. – 176 с.
3. Рейнгольд Э., Нивергельт Ю., Део Н. Комбинаторные алгоритмы теория и практика. – М.: Мир, 1980. – 476 с.
4. Липский В. Комбинаторика для программистов. – М.: Мир, 1988. – 200 с.
5. Тимошевская Н.Е. Параллельная генерация сочетаний и перестановок // Вторая Сибирская школа-семинар по параллельным вычислениям. – Томск: Изд-во Том. ун-та. – 2004. – С. 55–59.
6. Тимошевская Н.Е. О нумерации перестановок и сочетаний для организации параллельных вычислений в задачах проектирования управляющих систем // Известия Томского политехнического университета – Томск: Изд-во Томского политехнического университета, 2004.
7. Grama A., Kumar V. A survey of parallel search algorithms for discrete optimization problems // ORSA Journal of Computing. − 1995. – Vol. 7(4). − P. 365–385.
8. Семенов В.В. Алгоритмы параллельного перечисления разбиений множества // Тезисы Третьей Сибирской школы-семинара по параллельным вычислениям. – Томск: Изд-во Том. ун-та, 2005. – С. 29–30.
9. Ranade A.G.. Optimal speedup for backtrack search on a butterfly network // Proceedings of the Third ACM Symposium on Parallel Algorithms and Architectures, 1991.
10. Wei Shu, Kale L.V. A dynamic scheduling strategy for the share-kernel system // Proceedings of SuperComputing Conference. − 1989. – P. 389–398.
11. Furuichi M., Taki K. and Ichiyoshi N. A multi-level load balancing scheme for OR-parallel exhaustive search programs on the Multi-PSI // Proceedings of the Second ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. − 1990. – P. 50–59.
12. Тимошевская Н.Е. Параллельные методы обхода дерева // Математическое моделирование. − 2004. – Т. 16, № 1. – С. 105–114.
13. Grama A., Kumar V. Parallel processing for discrete optimization problem // Encyclopedia of microcomputers. − 1992. – P. 129–157.
14. Sanders P. Parallelizing NP-complete problems using tree shaped computations // Max-Planck-Institut fur Informatik, Im Stadtwald, Saarbrucken, Germany, 1999.
15. Gao Y., Marsland T.A. Multithreaded Pruned Tree Search in Distributed Systems // Computing Science Department University of Alberta Edmonton, Canada T6G 2H1.

16. Rao N.A., Kumar V. On the Efficiency of Parallel Backtracking // IEEE Trans. Parallel Distrib. Syst. − 1993. – Vol. 4(4). − P. 427–437.
17. Sanders P. Better Algorithms for Parallel Backtracking // Workshop on Algorithms for Irregularly Structured Problems, number 980 in LNCS. – 1995. – P. 333–347.
18. Rao V.N., Kumar V. Parallel depth-first search. Part I: Implementation // International Journal of Parallel Programming. − 1987. – Vol. 16(6). − P. 479–499.
19. Reinefeld A., Schnecke A. Work-Load Balancing in Highly Parallel Depth-First Search // Proceedings Scalable High Performance Computations Conference SHPCC'94. – 1994. – P. 773–780.
20. Kumar V., Rao V.N. Parallel depth-first search. Part II: Analysis // International Journal of Parallel Programming. − 1987. – Vol. 16(6). − P. 501–519.
21. Kryukova S. A. Parallel programming archetypes in combinatorics and optimization // Partial fulfillment of the Requirements for the Degree of master of Science. California Institute of Technology Computer Science department. – Pasadena, 1995.
22. Kumar V., Grama A., Rao V.N. Scalable load balancing techniques for parallel computers // Journal of Parallel and Distributed Computing. − 1994. – Vol. 22(1). − P. 60–79.
23. Arvindam S., Kumar V., Rao V.N. Floorplan optimization on multiprocessors // Proceedings of the 1989 International Conference on Computer Design. – 1989.
24. Saletore V., Kale L.V. Consistent linear speedup to a first solution in parallel state-space search // Proceedings of the 1990 National Conference on Artificial Intelligence. − 1990. – P. 227–233.
25. Sinha A.B., Kale L.V. A load balancing strategy for prioritized execution of tasks // Department of Computer Science University of Illinois Urbana, IL 61801. – 1992.
26. Korf R.E. Depth-first iterative-deepening: An optimal admissible tree search // Artificial Intelligence. − 1985. – Vol. 27. − P. 97–109.
27. Quinnand M.J., Deo N. An upper bound for the speedup of parallel branch-and-bound algorithms // BIT. − 1986. – Vol. 26, № 1.
28. Ferguson Ch., Korf R. Distributed tree search and its application to alpha-beta pruning // Proceedings of the 1988 National Conference on Artificial Intelligence, – 1988.
29. Finkel R.A., Manber U. DIB – a distributed implementation of backtracking // ACM Transactions of Programming Languages and Systems. − 1987. – Vol. 9, № 2. − P. 235–256.
30. Lai T.H., Sahni S. Anomalies in parallel branch and bound algorithms // Communications of the ACM. – 1984.

31. Li G.-J., Wah B.W.. Coping with anomalies in parallel branch-and-bound algorithms // IEEE Transactions on Computers. – 1986.
32. De Mello J.D., Calvet J.L., Garcia J.M. Vectorization and multitasking of dynamic programming in control: experiments on a CRAY-2 // Parallel Computing. − 1990. – Vol. 13. − P. 261–269.
33. Lee J., Shragowitz E., Sahni S. A hypercube algorithm for the 0/1 knapsack problem // Journal of Parallel and Distributed Computing. – 1988.
34. Ibarra O.H., Pong T.C., Sohn S.M. Parallel recognition and parsing on the hypercube // IEEE Transactions on Computers. − 1991. – Vol. 40(6). − P. 764–770.
35. Забиняко Г.И. Параллельный алгоритм целочисленной линейной оптимизации // Автометрия. − 2001. − № 6. − С. 107–114.

АЛГОРИТМЫ ПАРАЛЛЕЛЬНОГО ПЕРЕЧИСЛЕНИЯ РАЗБИЕНИЙ МНОЖЕСТВА

В. В. Семенов

Томский государственный университет

Рассматриваются задача перечисления всевозможных разбиений n-элементного множества и задача перечисления всех разбиений n-элементного множества на k блоков. Эти задачи возникают, например, при построении бесповторных ДНФ булевых функций. Предлагаются два параллельных алгоритма решения первой задачи и параллельный алгоритм решения второй. Каждый алгоритм обеспечивает равномерную загрузку процессоров при условии полного отсутствия взаимодействия между ними.

В первом алгоритме, рассчитанном на применение m процессоров, i-й процессор ($i = 1, \ldots, m$) перечисляет все разбиения на k блоков для $k_{i-1} < k \le k_i$, где $0 = k_0 < k_1 < \ldots < k_m = n$. Разбиения перечисляются каждым процессором с помощью последовательного алгоритма, представленного в [1]. Экспериментальным путем доказано (см. табл. 1), что наиболее равномерная загрузка процессоров и, следовательно, максимальная эффективность их использования таким алгоритмом достигаются при $m = 4$, $k_1 = [n/2] - 2$, $k_2 = [n/2] - 1$, $k_3 = [n/2]$. Из табл. 2 видно, что 4-й процесс работает значительно дольше остальных.

Таблица 1

Время работы (в секундах) последовательного алгоритма перечисления k-блочных разбиений n-элементного множества

$n\backslash k$	3	4	5	6	7	8	9	10	11
12	0	15	15	31	16	0	0	0	0
13	0	46	156	187	125	47	0	0	0
14	31	203	859	1250	1000	469	140	32	0
15	63	828	4328	7969	7843	4484	1532	343	47

Таблица 2

Время работы каждого из четырех процессов в первом алгоритме для множеств мощности 14, 15 и 16

$m\backslash n$	14	15	16
1	1,6	13,9	83,5
2	2,7	16,5	163,5
3	2,8	29,6	227,7
4	5,9	51,6	275,6

Во втором алгоритме – параллельного перечисления всевозможных разбиений n-элементного множества – используется дерево разбиений (рис. 1), в котором вершины i-го яруса, $i = 1, \ldots, n$ представляют все разбиения i-элементного множества. Потомки вершины v (вершины следующего яруса, смежные с v), сопоставленной разбиению R, получаются либо добавлением нового элемента x в один из блоков разбиения R, либо добавлением в R блока $\{x\}$.

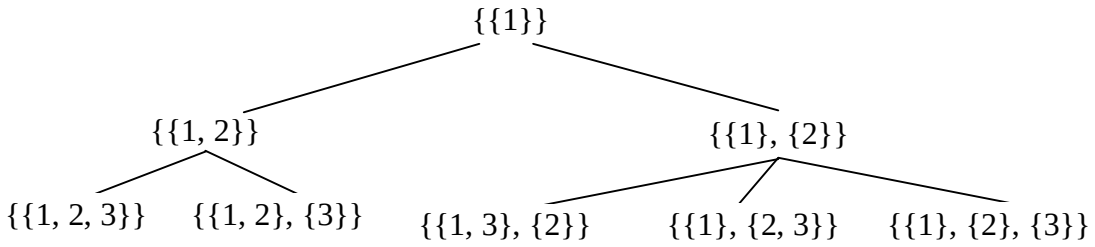


Рис. 1. Дерево разбиений множества $A = \{1, 2, 3\}$

Для параллельного перечисления разбиений используется параллельный обход дерева разбиений. На вход алгоритма подаются разбиваемое множество и номер яруса (*яруса назначений*), начиная с которого процессы, запущенные на разных процессорах, будут выполнять обход различных поддеревьев в дереве разбиений. Множество вершин яруса назначений разбивается на классы по числу процессов. Каждый из процессов поочередно выполняет обход поддеревьев с корневыми вершинами в назначенном ему классе.

Опишем действия, исполняемые каждым из m процессов с номерами $0, 1, \ldots, m–1$. Пусть t – номер яруса назначений. По последовательному алгоритму процесс перечисляет и нумерует разбиения t-элементного множества числами $0, 1, 2, \ldots$ до тех пор, пока не получит разбиение, номер которого сравним с номером процесса по модулю m. Тогда процесс выполняет обход поддерева с корнем в достигнутой вершине. После чего продолжает генерировать разбиения t-элементного множества для получения следующей корневой вершины своего класса.

Третий алгоритм предназначен для параллельного перечисления k-блочных разбиений n-элементного множества $A = \{a_1, \ldots, a_n\}$. Он отличается от второго алгоритма лишь структурой используемого дерева (рис. 2). В этом дереве потомки вершины v i-го яруса, соответствующей разбиению $R = \{R_1, R_2, \ldots, R_t\}$ множества $\{a_1, \ldots, a_i\}$, строятся по следующим правилам.

1. Если $(n – i) = (k – t)$, то единственный потомок вершины v соответствует разбиению $\{R_1, R_2, \ldots, R_t, \{a_{i+1}\}\}$, содержащему элемент a_{i+1} в отдельном блоке.

2. Если $t = k$, то вершина v имеет t потомков, которые сопоставляются следующим разбиениям множества $A_{i+1}=\{a_1, \ldots, a_{i+1}\}$: $\{R_1 \cup \{a_{i+1}\}, R_2, \ldots, R_t\}$, $\{R_1, R_2 \cup \{a_{i+1}\}, \ldots, R_t\}$, …, $\{R_1, R_2, \ldots, R_t \cup \{a_{i+1}\}\}$.

3. В остальных случаях вершина v имеет $(t+1)$ потомков, сопоставленных следующим разбиениям множества A_{i+1}: $\{R_1 \cup \{a_{i+1}\}, R_2, \ldots, R_t\}$, $\{R_1, R_2 \cup \{a_{i+1}\}, \ldots, R_t\}$, …, $\{R_1, R_2, \ldots, R_t \cup \{a_{i+1}\}\}$, $\{R_1, R_2, \ldots, R_t, \{a_{i+1}\}\}$.

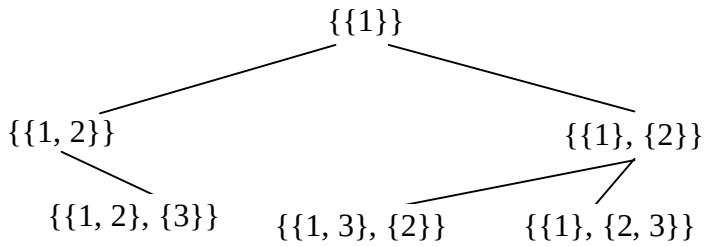


Рис. 2. Дерево разбиений множества
$A = \{1, 2, 3\}$ на 2 блока

Эффективность второго и третьего алгоритмов зависит от выбора номера яруса назначений. Очевидно, что чем больше номер яруса, тем меньшего размера поддеревья будут обрабатываться процессами и тем более равномерно будут загружены все процессоры. Однако если номер яруса взять слишком большим, то большая часть времени уйдет на построение корневых вершин, так как их построение ведется с помощью последовательного алгоритма, и, кроме того, каждый из процессов в итоге строит все разбиения, расположенные на ярусе назначений.

Таблица 3

Ускорение в алгоритме параллельного перечисления всевозможных разбиений *n*-элементного множества при *n* = 14, 15 и 16

n\s	2	3	4	5	6	7	8	9	10	11	12
14	1,99	2,99	3,99	4,96	5,95	6,96	7,91	8,96	9,88	10,84	11,79
15	1,99	2,99	3,98	4,95	5,93	6,96	7,87	8,95	9,85	10,86	11,74
16	1,99	2,98	3,97	4,92	5,92	6,95	7,83	8,93	9,80	10,84	11,68

Данные алгоритмы реализованы с помощью технологии *MPI* и экспериментально исследованы на кластерной системе с использованием от 2 до 14 процессоров. Для второго и третьего алгоритмов экспериментальным путем удалось подобрать такой номер яруса назначений, при котором эффективность распараллеливания близка к 1 (табл. 3, 4). Например, для множеств мощности n = 14, 15, 16 оптимальный номер яруса назначений равен 9 независимо от числа процессоров в системе.

Таблица 4

Ускорение в алгоритме параллельного перечисления *k*-блочных разбиений *n*-элементного множества при *n* = 15 и *k* = 4, 5, 6, 7 и 8

*k**s*	2	3	4	5	6	7	8	9	10	11	12
4	1,99	2,99	3,99	4,89	5,94	6,89	7,77	8,66	9,47	10,45	11,22
5	1,99	2,99	3,99	4,99	5,94	6,95	7,7	8,84	9,81	10,85	11,64
6	1,99	2,99	3,99	4,99	5,97	6,95	7,97	8,94	9,9	10,87	11,8
7	1,99	2,98	3,99	4,97	5,88	6,98	7,94	8,94	9,78	10,8	11,54
8	1,99	2,99	3,98	4,94	5,85	6,87	7,81	8,88	9,68	10,65	11,47

Достоинством представленных алгоритмов является отсутствие взаимодействия между процессорами, благодаря чему не тратится время на передачу данных, за исключением исходных. Кроме того, второй и третий алгоритмы являются хорошо масштабируемыми и позволяют достичь линейного ускорения.

Литература

1. Липский В. Комбинаторика для программистов. – М.: Мир, 1988. – 200 с.

МЕТОДЫ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ В ЗАДАЧАХ МОНИТОРИНГА КАЧЕСТВА АТМОСФЕРНОГО ВОЗДУХА*

Д.А. Беликов

Томский государственный университет

Мониторинг и прогноз уровня загрязнения атмосферы города являются одной из актуальных задач охраны окружающей среды, для решения которой успешно применяются методы математического моделирования. Область применения математических моделей постоянно расширяется и на данный момент включает следующие задачи: выявление характерных особенностей распространения загрязнений над выбранной территорией при различных погодных условиях, изучение вклада отдельных источников в общий баланс загрязнения атмосферы, оценку последствий возможных аварийных ситуаций на предприятиях повышенной опасности, оценку воздействия проектируемых объектов и т.п.

Чтобы достоверно прогнозировать загрязнение воздуха, необходимо учитывать химические и фотохимические реакции, происходящие в атмосфере между продуктами первичной эмиссии (выбросами промышленных предприятий) с участием компонентов воздуха и приводящие к образованию новых соединений, многие из которых весьма токсичны (озон, перацетилнитраты). С этой целью разрабатываются и внедряются в модели переноса примеси механизмы химических реакций. К наиболее известным и широко используемым кинетическим схемам химических реакций относятся:

CB4 (Carbon Bond 4) [1]. В базовом варианте включает 36 компонентов и 93 реакции (11 из них фотохимические). Доступны расширенные варианты этого механизма, учитывающие реакции с участием атмосферной влаги и реакции образования аэрозолей. Механизм CB4 компилируется с различными моделями переноса примеси (EPA’s Empirical Kinetic Modeling Approach, Urban Airshed Model IV и V [1], CAMx [4]).

RADM2 (Regional Acid Deposition Mechanism version 2) [2], разработанная в 1990 г. В базовом варианте включает 57 компонентов и 158 реакций (21 из них фотохимические). Механизм RADM2 используется в моделях предсказания качества воздуха CMAQ [3].

На основе RADM2 разработан современный механизм RACM (Regional Atmospheric Chemistry Mechanism), в общих чертах схожий с

* Работа выполнена при финансовой поддержке РФФИ, гранты № 04-07-90219, № 05-05-98010р_обь.

RADM2 и включающий 77 компонентов и 237 уравнений. Механизм используется в модели предсказания качества воздуха EURAD [5].

В данных кинетических схемах химических реакций только некоторые соединения рассчитываются явно, остальные представляются посредством классов, отобранных в зависимости от типа связи атомов углерода в молекуле, или от молекулярного веса, или по какому-либо другому принципу. Такое неявное представление химических компонентов продиктовано необходимостью создания эффективных методов для решения системы уравнений химической кинетики.

В большинстве случаев применение кинетических схем с большим набором реакций позволяет на высоком уровне обеспечить воспроизведение процессов генерации и трансформации веществ в атмосфере. Однако кроме количества рассматриваемых химических реакций и числа реагирующих компонентов примеси, что, в первую очередь, определяет полноту представления в модели реально протекающих химических взаимодействий, существенным фактором является возможность явно рассчитывать концентрации отдельных соединений. Не последнее значение имеет и скорость вычислений. Подробные схемы при более тщательном моделировании атмосферной химии значительно уступают в быстродействии упрощенным механизмам, которые при менее детальном представлении химических процессов позволяют с приемлемой точностью явно рассчитывать концентрации соединений. Заметим, что качество расчетов по модели химических взаимодействий в значительной степени зависит от точности задания начальных и фоновых условий, а также данных о первичной эмиссии примеси, определение которых представляет собой непростую задачу.

Для оценки и прогнозирования загрязнения атмосферы тропосферным озоном и другими вторичными загрязнителями может быть использована упрощенная полуэмпирическая схема GRS (Generic Reaction Set) [6], которая содержит 11 химических компонентов (реагирующая часть смога R_{smog}, органические радикалы RP, перекись водорода H_2O_2, оксид азота NO, диоксид азота NO_2, озон O_3, диоксид серы SO_2, устойчивые негазообразные органические соединения углерода $SNGOC$, устойчивые газообразные соединения азота SGN, устойчивые негазообразные соединения азота $SNGN$, устойчивые негазообразные соединения серы $SNGS$), участвующих в 8 химических реакциях (таблица).

Данная схема включена в трехмерную негидростатическую модель TAPM (The Air Pollution Model) [6], которая представляет собой программный комплекс, нацеленный на изучение мезомасштабной метеорологии, а также для исследования переноса и трансформации загрязнителей. Для верификации модели TAPM был проведен ряд сопоставлений результатов расчетов с данными измерений по метеорологии и качеству воздуха. Оценка моделирования образования вторичных загрязнителей проводилась в реальных условиях города Мельбурна и области Пилбара, что на севере

Австралии. Полученные результаты свидетельствуют о достижении высокого качественного и количественного уровня воспроизведения полей метеорологических параметров и полей концентрации озона, оксидов азота и аэрозолей [6].

Реакции и скорости реакций фотохимической схемы GRS

Реакции
Скорости реакций
Значения коэффициентов k_i

$$\mathrm{R}_{smog} + h\nu \rightarrow \mathrm{RP} + \mathrm{R}_{smog} + \eta\mathrm{SNGOC}$$
$$r_1 = k_1 \mathrm{C}_{Rsmog}$$
$$k_1 = k_3 e^{(-4700(1/T - 1/316))}$$

$$\mathrm{RP} + \mathrm{NO} \rightarrow \mathrm{NO}_2$$
$$r_2 = k_2 \mathrm{C}_{\mathrm{RP}} \mathrm{C}_{\mathrm{NO}}$$
$$k_2 = 3580/60T$$

$$\mathrm{NO}_2 + h\nu \rightarrow \mathrm{NO} + \mathrm{O}_3$$
$$r_3 = k_3 \mathrm{C}_{\mathrm{NO}_2}$$
$$k_3 = 0.0001\delta \cdot TSR/60$$

$$\mathrm{NO} + \mathrm{O}_3 \rightarrow \mathrm{NO}_2$$
$$r_4 = k_4 \mathrm{C}_{\mathrm{NO}} \mathrm{C}_{\mathrm{O}_3}$$
$$k_4 = (924/60T)e^{-1450/T}$$

$$\mathrm{RP} + \mathrm{RP} \rightarrow \mathrm{RP} + \alpha\mathrm{H}_2\mathrm{O}_2$$
$$r_5 = k_5 \mathrm{C}_{\mathrm{RP}} \mathrm{C}_{\mathrm{RP}}$$
$$k_5 = 1000/60$$

$$\mathrm{RP} + \mathrm{NO}_2 \rightarrow \mathrm{SGN}$$
$$r_6 = k_6 \mathrm{C}_{\mathrm{RP}} \mathrm{C}_{\mathrm{NO}_2}$$
$$k_6 = 0.12/60$$

$$\mathrm{RP} + \mathrm{NO}_2 \rightarrow \mathrm{SNGN}$$

$$r_7 = k_7 \mathrm{C}_{\mathrm{RP}} \mathrm{C}_{\mathrm{NO}_2}$$
$$k_7 = k_6$$

$$\mathrm{RP} + \mathrm{SO}_2 \rightarrow \mathrm{SNGS}$$
$$r_8 = k_8 \mathrm{C}_{\mathrm{RP}} \mathrm{C}_{\mathrm{SO}_2}$$
$$k_8 = 0.1 k_6$$

Эйлерова модель турбулентной диффузии, включающая транспортные уравнения с описанием адвекции, турбулентной диффузии и химических реакций схемы GRS, представима в виде [6]

$$\frac{\partial C_i}{\partial t} + \frac{\partial UC_i}{\partial x} + \frac{\partial VC_i}{\partial y} + \frac{\partial W_c C_i}{\partial z} = -\frac{\partial}{\partial x}\langle c_i u\rangle - \frac{\partial}{\partial y}\langle c_i v\rangle - \frac{\partial}{\partial z}\langle c_i w_c\rangle + S_i + R_i; \quad (1)$$
$$i = 1,..,n;$$

здесь $C_i(t,x,y,z)$ – концентрация i-го компонента примеси; W_c – вертикальная составляющая скорости примеси; S_i – источниковый член, представляющий выбросы компонентов примеси в атмосферу и их осаждение на подстилающую поверхность; R_i описывает образование и трансформацию вещества за счет химических реакций с участием компонентов примеси; n – количество химических компонентов примеси, принятых к рассмотрению в схеме химических реакций.

Уравнения изменения концентрации компонентов примеси для схемы GRS имеют вид

$$\begin{aligned}
&R_{[\mathrm{APM}]} = F_{\mathrm{CH}_2}\eta r_1 + F_{\mathrm{HNO}_3} r_7 + F_{\mathrm{H_2SO_4}}(r_8); && R_{[\mathrm{SP_x}]} = r_2 - r_6 - r_7;\\
&R_{[\mathrm{RP}]} = r_1 - r_2 - r_5 - r_6 - r_7 - r_8; && R_{[\mathrm{H_2O_2}]} = \sigma r_5;\\
&R_{[\mathrm{SO_2}]} = -r_8; && R_{[\mathrm{NO_x}]} = -r_6 - r_7;\\
&R_{[\mathrm{NO_2}]} = r_2 + r_4 - r_3 - r_6 - r_7; && R_{[\mathrm{R_{smog}}]} = 0,
\end{aligned} \quad (2)$$

где APM – класс аэрозолей, объединяющий собой устойчивые негазообразные органические соединения углерода SNGOC, устойчивые негазообразные соединения азота SNGN и серы SNGS; $[\mathrm{NO_x}]=[\mathrm{NO}]+[\mathrm{NO_2}]$ и $[\mathrm{SP_x}]=[\mathrm{O_3}]+[\mathrm{NO_2}]$ – суммарные концентрации, введенные для обеспечения более высокой скорости получения решения системы уравнений химической кинетики, $F_{\mathrm{CH_2}}$=0,57, $F_{\mathrm{H_2SO_4}} = 4,0$, $F_{\mathrm{H_2SO_4}} = 4,0$ – приближенные коэффи-

циенты, определяющие переход устойчивых негазообразных соединений в аэрозоли, $\eta = 0{,}1$, $\alpha = \max\left(0{,}03, \exp\left(-0{,}0261 \frac{[R_{smog}]}{[NO_x]}\right)\right)$ – численные коэффициенты [6].

Входящие в уравнения (1) осредненные компоненты скорости ветра U, V определяются на основе модели АПС [7].

Для определения неизвестных корреляций $\langle c_i u \rangle$, $\langle c_i v \rangle$, $\langle c_i w \rangle$ в уравнениях (1), представляющих турбулентную диффузию примеси, используются дополнительные выражения в виде соотношений Буссинеска [7,8] или алгебраических выражений [9,10].

Пространственные нестационарные уравнения (1) решаются численно конечно-разностным методом в области с разнообразными источниками выбросов на подстилающей поверхности и высотными точечными источниками. Аппроксимация дифференциальных операторов осуществлена со вторым порядком точности по координатам и первым – по времени с использованием явных разностных схем для всех членов уравнения за исключением вертикальной диффузии. Такой способ дискретизации дифференциальной задачи позволяет решить проблемы, возникающие из-за нелинейности уравнений, и существенно ускорить получение результата вследствие применения экономичного метода прогонки при приемлемом ограничении на величину шага по времени [11]. Адвективные члены уравнений переноса (1) аппроксимируются с использованием монотонизированной противопотоковой схемы Ван Лира. Более подробно метод решения, используемая сетка, начальные и граничные условия описаны в [12].

Учет химических взаимодействий в модели требует расчета каждого компонента примеси из соответствующих уравнений химических реакций, решение которых весьма трудоёмко и затратно. Поэтому такие модели нуждаются в привлечении ЭВМ с высокой производительностью. Эффективным способом сокращения времени расчета, активно осваиваемым в настоящее время, является применение вычислительных систем с параллельной архитектурой. Принцип работы таких ЭВМ основан на том, что некоторые части вычислительного алгоритма могут выполняться независимо друг от друга, то есть параллельно [13]. В случае, если доля независимых, т.е. подлежащих распараллеливанию частей алгоритма велика, ускорение параллельной реализации алгоритма может оказаться значительным.

В настоящее время наибольшее распространение получили вычислительные системы, построенные с использованием кластерных технологий. Системы такого типа легко масштабируются, отличаются дешевизной и простотой проектирования. В 26-й редакции списка 500 мощнейших компьютеров мира Top500 (http://top500.org/lists/2005/11/), опубликованной в ноябре 2005 г., 360 систем или более 72% являются кластерными.

Дешевизна и простота архитектуры подобных систем оборачиваются сложностью обеспечения взаимодействия параллельных процессов между собой, потому что для обмена данными между процессами в кластерных системах необходимо производить передачу данных между узлами кластера. Время, затрачиваемое на обеспечение одной передачи данных, значительно превосходит время одной арифметической операции с этими данными: отношение времен может достигать тысяч и даже миллионов раз. Следовательно, для получения высокоэффективной параллельной реализации необходимо произвести согласованное распределение вычислений и данных между процессорами крупными порциями, а также обеспечить каждому процессору доступ к удаленным данным – данным, с которыми оперирует другой процессор, сведя при этом количество коммуникационных обменов к минимуму.

Декомпозиция расчетной области при моделировании пространственных процессов в атмосфере может быть проведена тремя способами: декомпозиция по трем направлениям (вертикальному и двум горизонтальным), двумерная декомпозиция (по двум направлениям) и одномерная. Для выбранного численного метода декомпозиция данных по вертикальной координате представляется нежелательной из-за применения прогонки в вертикальных столбцах сетки. Этот метод плохо распараллеливается для систем с распределенной памятью вследствие больших затрат на межпроцессорные обмены, которые свойственны моделям атмосферного пограничного слоя и моделям переноса примеси.

При небольшом числе используемых процессоров эффективность параллельных программ, построенных с применением различных способов декомпозиции области исследования, примерно одинакова, поэтому выбрана более простая одномерная декомпозиция по горизонтальной координате X. Схема декомпозиции представлена на рис. 1.

Отметим, что декомпозиция вдоль обеих горизонтальных координат предпочтительнее при наличии вычислительных систем с большим числом процессоров.

Проведём теоретическую оценку ускорения параллельной версии программы, выполняющейся на p узлах по отношению к однопроцессорной реализации. Примем за t_a время одной арифметической операции, а t_o – время для передачи одного числа между процессами. Пусть расчетная сетка $M \times M \times N$ узлов. Тогда время, затраченное на нахождение решения дискретного аналога одного уравнения переноса (1) на одном шаге по времени для последовательного варианта программы, можно оценить как $T_1 = m \cdot N \cdot M \cdot M \cdot t_a$; m – число арифметических операций метода при вычислении одного значения сеточной функции.

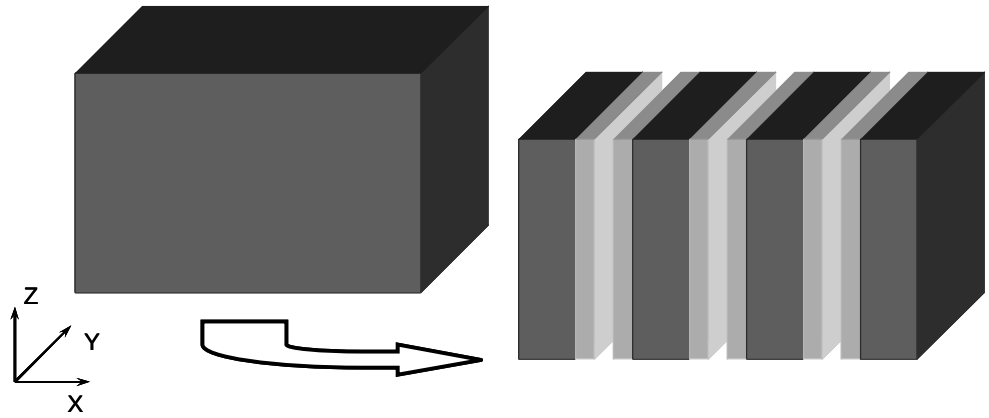


Рис. 1. Схема декомпозиции расчетной области. Светло-серым цветом обозначены области, данные из которых участвуют в обменных операциях

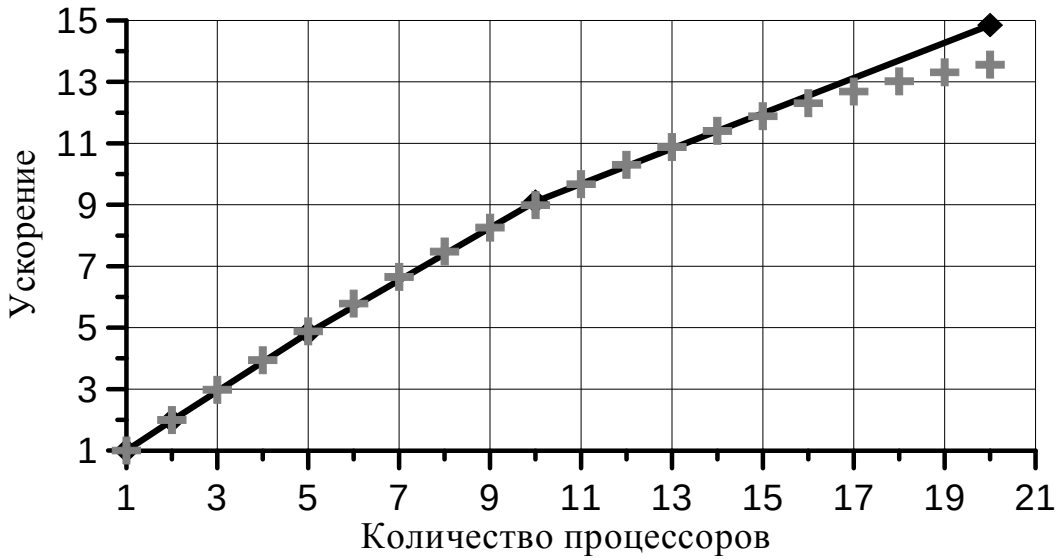


Рис. 2. Оценка ускорения параллельной реализации программы. Символы представляют результаты теоретической оценки, черная линия – расчеты на кластере ИОА СО РАН

На p процессорах время счета складывается из времени выполнения арифметических операций T_p^a и времени передачи данных между процессами T_p^o:

$$T_p = T_p^a + T_p^o = m \cdot N \cdot M \cdot M \cdot \frac{t_a}{p} + 4N \cdot M \cdot t_o (p-1).$$

Тогда ускорение S_p будет

$$S_p = \frac{T_1}{T_p} = p \Big/ \left(1 + \frac{4\kappa(p-1)p}{mM}\right), \text{ где } \quad \kappa = \frac{t_o}{t_a}.$$

Результаты сравнения теоретической оценки ускорения и практических расчетов на кластере ИОА СО РАН представлены на рис. 2. Как видно, кривая и символы почти сливаются, следовательно, практическая реализация параллельной версии расчетного алгоритма выполнена правильно.

Полученное пятнадцатикратное ускорение на 20 процессорах говорит о хорошей степени параллелизма алгоритма и удачном планировании обменов между процессами. Заметим, что достаточно высокое ускорение обусловлено значительными размерами расчетных подобластей и трудоемкостью вычислений в каждой из них. Поэтому доля обменов между процессами составляет незначительную часть от общего времени выполнения программы.

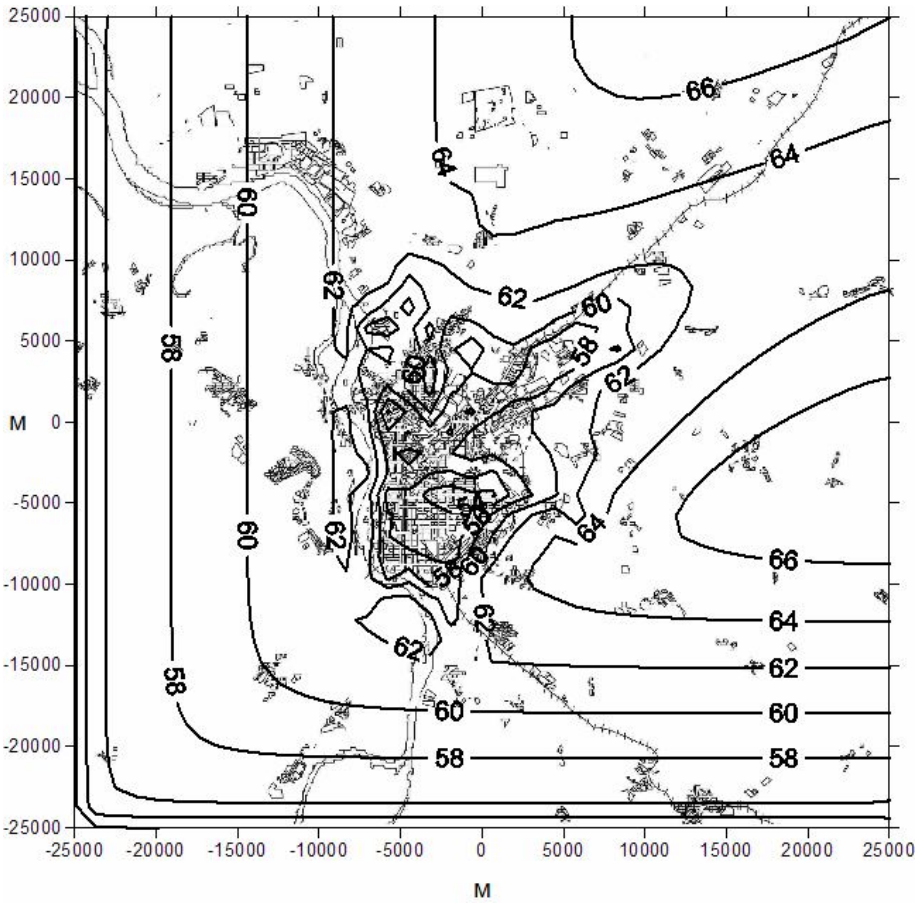


Рис. 3. Распределение концентрации озона (мкг/м^3) над городом Томском и пригородами на высоте 10 м. Расчет соответствует 14:00 часам 16 октября 2003 г.

Предложенная модель переноса примеси с учетом химических реакций позволяет исследовать динамику распределения приземных концентраций примесей первичной и вторичной эмиссий [112]. На рис. 3 представлено полученное расчетным путем распределение концентрации загрязнителя – тропосферного озона над городом Томском и пригородами на 14:00 16 октября 2003 г.

В работе проведен обзор механизмов химической кинетики, используемых для исследования химических реакций в атмосфере, и методов декомпозиции расчетной области при применении вычислительных систем с параллельной архитектурой. Представлена модель прогноза качества воздуха с учетом протекающих в атмосфере химических взаимодействий, для которой получена теоретическая и расчетная оценки ускорения параллельной версии программы. Выявлено, что весьма эффективно использование вычислительных систем с параллельной архитектурой для исследования уровня загрязнения атмосферы с учетом химических трансформаций загрязнителей. Представлены результаты исследования загрязнения воздуха в Томске.

Литература

1. Gery M. W., Whitten G.Z., Killus J.P., Dodge M. C. A Photochemical Mechanism for Urban and Regional Scale Computer Modeling // Journal of the Geophysics Research. − 1989. − Vol. 94. − P. 12925–12956.
2. Stockwell W.R., Kirchner F., Kuhn M. and Seefeld S. A New Mechanism for Atmospheric Chemistry Modeling // Journal of Geophysical Research. − 1997. − Vol. 102, № 22. − P. 25847–25879.
3. Byun D.W., Ching J.K.S. Science Algorithms of the EPA Models-3 Community Multiscale Air Quality (CMAQ) Modeling System // EPA Report N. EPA-600/R-99/030, Office of Research and Development. U.S. Environmental Protection Agency, Wachington, Dc., 1999. – 11 p.
4. www.camx.com/files/CAMxUsersGuide_v4_20.pdf
5. Elbern H., Schmidt H. Ozone episode analysis by four-dimensional variation chemistry data assimilation // Journal of the Geophysics Research. − 2001. − Vol. 106. − P. 3569–3590.
6. Hurley P. J. The Air Pollution Model (TAPM) Version 2 // CSIRO Atmospheric Research Technical Paper. – 2002. – №55. – 39 p.
7. Старченко А.В., Беликов Д.А. Численная модель для оперативного контроля уровня загрязнения городского воздуха // Оптика атмосферы и океана. − 2003. – Т.16, №7. − С. 657–665.
8. Берлянд М.Е. Прогноз и регулирование загрязнений атмосферы. – Л.: Гидрометеоиздат, 1985. – 271 с.
9. Беликов Д.А., Старченко А.В. Моделирование турбулентного переноса примеси в пограничном слое атмосферы // Исследования по баллистике и смежным вопросам механики. – Томск: Изд-во Том. ун-та, 2002. – Вып.5. – С. 67–68.
10. Старченко А.В. Моделирование переноса примеси в однородном атмосферном пограничном слое // Труды Международной конференции ENVIROMIS-2000. – Томск: ЦНТИ, 2000. – С. 77–82.
11. Самарский А.А., Николаев Е.С. Методы решения сеточных уравнений. − М.: Наука, 1978. – 591 с.

12. Беликов Д.А., Старченко А.В. Исследование образования вторичных загрязнителей (озона) в атмосфере г. Томска // Оптика атмосферы и океана. − 2005. – Т. 18, № 5-6. − С. 435–443.
13. Старченко А.В., Есаулов А.О. Параллельные вычисления на многопроцессорных вычислительных системах. – Томск: Изд-во Том. ун-та, 2002. – 56 с.

ПРОГНОЗИРОВАНИЕ КОЛИЧЕСТВА ЛЕСНЫХ ПОЖАРОВ И ЛАНДШАФТНОЕ РАСПАРАЛЛЕЛИВАНИЕ

Н.В. Барановский

Томский государственный университет

Введение

Прогноз лесной пожарной опасности является очень актуальной проблемой. Анализ существующих в России и за рубежом методик прогноза лесной пожарной опасности показывает, что практически все методики имеют под собой слабую физическую основу и, как правило, учитывают только метеоданные. Грозовая активность и антропогенная нагрузка должным образом не учитываются. В аналитическом обзоре [1] представлены последние данные – в России 1807 лесхозов, 7851 лесничество, в каждом лесничестве более 100 кварталов, в каждом квартале может быть до 100 выделов (однородных участков). Необходимо применять многопроцессорные вычислительные системы (МВС) и необходима разработка соответствующего проблемно-ориентированного подхода ландшафтного распараллеливания для прогноза лесной пожарной опасности на крупных лесопокрытых территориях, который бы базировался на трех основах: физически содержательная методика определения вероятности лесопожарного происшествия (в данном случае возникновение лесного пожара), использование достаточно дешевых МВС, существующая структура устройства лесного хозяйства.

1. Основные положения ландшафтного распараллеливания

Математическая постановка должна быть такой, чтобы расчеты для каждого отдельного однородного участка леса не зависели от других участков леса. Другими словами, необходимо разработать математическую постановку, которая бы позволяла вести расчет отдельно по всем выделам.

Однородный участок леса должен совпадать с однородными участками леса, принятыми в лесотаксационных описаниях [2]. Отсюда берет свое начало термин «ландшафтное распараллеливание».

Для обеспечения более или менее равномерной вычислительной загрузки процессорных узлов МВС необходимо использовать модель распараллеливания по данным [3]. В MPI, например, может быть использована SPMD-модель вычислений [4]. Хотя равномерная загрузка данными не всегда обеспечивает равномерность вычислительной загрузки. Интере-

сующиеся перебалансировкой вычислительной нагрузки процессорных узлов МВС могут обратиться к работе [5].

Возможность обеспечивать высокую производительность (быстродействие) параллельной программы на как можно большем количестве МВС, различных по архитектуре, имеющих при этом хорошее соотношение цена/производительность, доступность и массовое распространение.

Обеспечение как можно меньшего количества обменов информационными пакетами между ветвями параллельной программы в процессе ее выполнения, так как основное узкое место МВС – это коммуникационная среда, применяемая для связи между процессорными узлами. Таким образом, самым оптимальным вариантом по критерию цена/производительность можно считать кластерные вычислительные системы, имеющие, однако, очень низкие показатели скорости обмена между процессорными узлами. Это и обусловливает необходимость как можно меньшего числа обменов и обеспечения возможности перекрытия вычислений и обменов.

Обеспечение однозначного отображения административно-хозяйственного, лесотаксационного разделения лесного фонда государства (по крайней мере, Российской Федерации) на архитектуру МВС. Разбиение лесного фонда по выделам и распределение вычислительной нагрузки по процессорным узлам МВС.

2. Детерминированно-вероятностный подход определения числа лесных пожаров

Используя теорию вероятностей и физические соображения, получена формула для оценки вероятности возникновения лесного пожара для *j*-го временного интервала лесопожарного сезона [6]:

$$P_j = \sum_{i=1}^{N} \left[A_{ij} P_{ij}(A) P_{ij}(ЛП/A) + M_{ij} P_{ij}(M) P_{ij}(ЛП/M) \right] P_{ij}(C). \qquad (1)$$

Здесь P_j – вероятность возникновения лесного пожара для j-го интервала на контролируемой лесной территории; F – площадь лесной территории конкретного лесхоза, района или области; F_i – площадь лесной территории, покрытой лесом *i*-го типа (по лесотаксационным описаниям это соответствует территории отдельно взятого выдела); N – общее число выделов на площади F; A_{ij}, M_{ij} – весовые коэффициенты антропогенной нагрузки (динамика в течение недели) и грозовой активности (динамика наземных грозовых разрядов); $P_{ij}(A)$ – вероятность антропогенной нагрузки; $P_{ij}(ЛП/A)$ – вероятность возникновения пожара вследствие антропогенной нагрузки на территории выдела F_i ; $P_{ij}(M)$ – вероятность возникновения сухих гроз на территории выдела F_i; $P_{ij}(ЛП/M)$ – вероятность возникновения лесного пожара от молнии при условии, что сухие грозы могут иметь ме-

сто на территории выдела F_i; $P_{ij}(C)$ – вероятность возникновения пожара по метеоусловиям лесопожарного созревания; индекс j соответствует дню пожароопасного сезона.

Для определения всех членов в формуле (1) необходимо использовать определение вероятностей через частоту событий и воспользоваться статистическими данными для конкретного лесхоза [6]:

$$A_{ij} \approx \frac{N_{ПДН}}{N_{ПН}},\ P_{ij}(A) \approx \frac{N_A}{N_{ПС}},\ P_{ij}(ЛП/A) \approx \frac{N_{ПА}}{N_{КП}}; \quad (2)$$

$$M_{ij} \approx \frac{N_{МЧ}}{N_{МС}},\ \ P_{ij}(M) \approx \frac{N_M}{N_{ПС}},\ \ P_{ij}(ЛП/M) \approx \frac{N_{ПМ}}{N_{КП}}, \quad (3)$$

где N_A – количество дней в пожароопасный сезон, когда имеется антропогенная нагрузка, достаточная для зажигания ЛГМ; $N_{ПА}$ – количество пожаров от антропогенной нагрузки; $N_{КП}$ – общее количество пожаров; N_M – число дней, в которые имели место молнии (при сухих грозах); $N_{ПС}$ – общее число дней пожароопасного сезона; $N_{ПМ}$ – количество пожаров от молний при сухих грозах.

Число прогнозируемых лесных пожаров определяется по формуле

$$PNF(d) = \frac{RNF(d-1)P_j^0(d)}{P_j^0(d-1)}. \quad (4)$$

Таким образом, число прогнозируемых пожаров на конкретный день зависит от числа зарегистрированных пожаров в предыдущий день, умноженное на отношение вероятностей возникновения лесных пожаров в текущий и предшествующий дни.

3. Параллельная реализация

3.1 Оценки ускорения и эффективности параллельных программ

Параллельная программа, которая реализует расчет времени сушки слоя ЛГМ на подстилающей поверхности и вероятности возникновения лесных пожаров на крупных территориях, состоит из одного одномерного цикла с индексом i и количеством итераций NV над структурой данных D.

По аналогии с работой [7] сначала рассмотрим выполнение однородного одномерного цикла на виртуальной вычислительной системе W, которую будем представлять себе в виде полного графа с неограниченным количеством вершин $w_i \in W$ (i = 1,2,...). При параллельном выполнении цикла компоненты структуры D, которые обрабатываются на разных итерациях цикла, размещаются на разных вершинах виртуальной вычислительной системы W. Пусть цикл вычисляет значение $F(D)$. В этом случае i-я итерация рассматриваемого цикла будет выполняться на вершине $w_i \in W$: вычисляется $F(d_i) = f_i(D_i)$, где $D_i \in D$, d_i − компонента структуры D, распределенная на вершину w_i, а функция f_i представляет операции, которые вы-

полняются на *i*-й итерации. Мы рассматриваем однородный цикл. Так как все f_i одинаковы, то будем обозначать их через *f*. Множества D_i зависят от структуры данных *D* и от алгоритма вычислений *F(D)*. D_i совпадает с d_i, так как данные независимы. Между вершинами не будет обменов.

В реальной ситуации выполнение параллельного цикла производится не на виртуальной, а на конкретной МВС. Данную систему можно представить в виде графа *G* с *M* узлами, где *M* − количество процессоров вычислительной системы. На узлах графа *G* задана вещественная функция *c* ($c(g_k) = c_k$), а на его дугах (парах узлов) – вещественная функция *e* ($e(g_k,g_j)=e_{kj}$). Значения этих функций на каждом узле или дуге определяются соответственно производительностью процессора, представленного этим узлом (количеством операций в единицу времени), и скоростью передачи данных по каналу связи между процессорами вычислительной системы (скорость передачи данных по каналу связи определяется его пропускной способностью, т.е. количеством байт, передаваемых в единицу времени, латентностью).

В случае, когда вычислительная система состоит из одного узла (*M* = 1), время выполнения цикла – $T_o=pNV/c$, где *p* − количество операций, необходимых для вычисления одной итерации цикла, *NV* − число итераций цикла, с – производительность узла. Так как вычислительная система однородна, то при любом *k* $c_k = c$, и для любых *k* и *j* e_{kj} = е. Тогда, если *NV* > *M* и *NV* делится нацело на М, на каждом процессоре выполняется *NV/M* итераций цикла, и время выполнения цикла равно $T_M=pNV/(Mc)$. Определим ускорение S_M через отношение T_0 к T_M и в итоге получим: $S_M=M$. Эта формула для ускорения справедлива при полном отсутствии потерь. Под эффективностью понимается отношение $E = \frac{T_0}{MT_M}$, которая в данном случае будет равна единице, так как не учитываются потери.

Если *NV* не делится нацело на *M*, то задача об оптимальном распределении итераций цикла по процессорным узлам системы является задачей целочисленного программирования и, как отмечено в [7], не всегда имеет точное решение. Рассмотрим здесь следующую эвристику: сначала [*NV/M*] итераций равномерно распределяются по узлам, затем количество оставшихся итераций распределяется на $NV–[NV/M]\cdot M$ процессорных узлов, что приведет к тому, что на некоторых узлах будет больше итераций.

Кроме того, перед выполнением цикла, как правило, необходимо разослать исходные данные по узлам вычислительной системы, а после выполнения цикла необходимо собрать результаты с процессорных узлов. Время, которое необходимо на выполнение указанных действий, обозначим через T'_M. Если *NV* не делится нацело на *M*, а при распределении данных по узлам была использована некоторая звристика, то на часть узлов попадут «лишние» данные [7], время обработки которых тоже присоеди-

ним к T'. В итоге время выполнения параллельного цикла равно $T_M=T^0_M+T'_M=T^0_M+\alpha_M T^0_M$, где T^0_M – время выполнения параллельного цикла при отсутствии потерь, а $\alpha_M=T'_M/T^0_M$. Таким образом, с учетом потерь ускорение равно $S_M=M/(1+\alpha_M)$. Когда необходимо учитывать потери, эффективность вычисляется по формуле $E=\frac{T_0}{MT_M}=\frac{T_0}{M\cdot\left(T^0_M+\alpha_M T^0_M\right)}$ и будет равна $1/(1+\alpha_M)$.

3.2 Результаты работы параллельной программы

Расчеты проводились на вычислительном кластере Томского государственного университета. Технические характеристики: 18 процессоров Pentium III 650 МГц, 2.5 Гб ОЗУ, 36 Гб дискового пространства, Fast Ethernet. Программное обеспечение: операционная система FreeBSD 4.2, кластерный пакет LAM MPI v 6.3.2. Кроме того, был создан и запущен в эксплуатацию вычислительный кластер на базе компьютеров локальной вычислительной сети кафедры [8], который состоит из 4 однопроцессорных компьютеров на базе процессоров Intel Pentium III 800 МГц. На компьютерах установлено по 128 Мб оперативной памяти и жесткий диск на 20 Гб. Компьютеры объединены с помощью сетевой технологии Ethernet. Для организации параллельных вычислений на кластере установлен кластерный пакет LAM MPI 6.5.1.Узлы кластера в одно и то же время могут использоваться как в качестве пользовательских станций, так и непосредственно для проведения вычислений.

Реализация параллельной MPI-программы осуществляется на языке программирования «Си» с применением функций библиотеки передачи сообщений MPI. Алгоритм параллельной программы: исходные данные разрезаются и рассылаются корневым процессом параллельной программы. Затем на каждом узле параллельная программа обрабатывает свою часть данных. Межпроцессорных обменов нет, распараллеливание организовано крупнозернистыми блоками. По окончании вычислений корневой процесс собирает результаты со всех процессорных узлов и сохраняет их в файле результатов.

Разработаны масштабируемые параллельные программы (число задействованных процессоров является параметром программы).

На рис. 1 представлены зависимости эффективности (1.a) и ускорения работы (1.b) параллельной программы от числа задействованных процессоров параллельной программы.

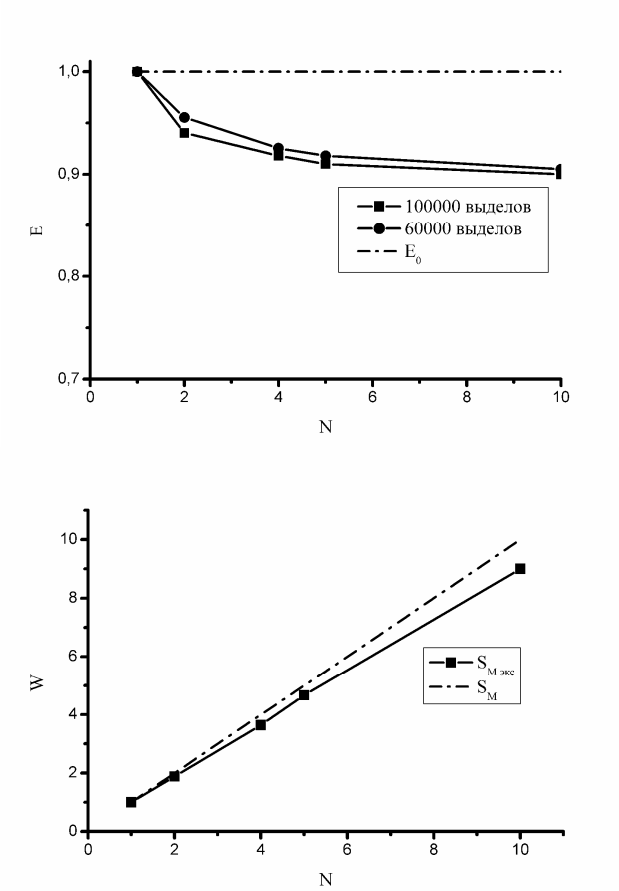


Рис. 1. Эффективность и ускорение
параллельной программы

Анализ результатов, представленных на рис. 1, показывает, что реальная эффективность по сравнению с оценкой падает незначительно. Разница в результатах, соответствующих 100000 и 60000 выделов, объясняется тем, что с ростом вычислительной нагрузки NV размер информационных пакетов, которые передаются по коммуникационной среде, также увеличивается.

В реальной ситуации ускорение несколько меньше, чем полученное при оценке без учета потерь, так как часть времени теряется на рассылку данных в начале и сбор результатов на завершающей стадии работы параллельной программы. Видно, что с ростом числа процессоров ускорение несколько падает, хотя вычислительная нагрузка на каждый процессор уменьшается. Однако в то же время увеличивается доля времени, которое затрачивает корневой процесс на взаимодействие с другими процессами при рассылке и сборе данных, так как скоростные показатели коммуникационной среды вычислительного кластера все-таки не велики и здесь начинают играть роль архитектурные аспекты.

Проанализировав полученные результаты, можно сделать вывод о том, что алгоритм рассматриваемой задачи очень хорошо подходит для па-

раллельной реализации на МВС. В результате имеет место ускорение, практически пропорциональное числу задействованных процессоров МВС. Эффективность практически не зависит от числа задействованных процессоров, так как в процессе работы программы нет межпроцессорных передач данных. Установлено, что падение производительности невелико и $\max_{M\in[1,16]}(\alpha_M)=0{,}09$.

Следует заметить, что несмотря на применение эвристики для распределения данных в случае, когда NV не делится нацело на M (статическая эвристика с предварительным распределением перед началом выполнения параллельного цикла), это не приводит к заметной разбалансировке вычислительной нагрузки и нет необходимости применять средства динамической перебалансировки вычислительной нагрузки в процессе выполнения параллельной программы. Рассылка и сбор данных корневым процессом параллельной программы технически реализованы посредством векторных вариантов функций рассылки и сбора данных библиотеки MPI.

4. Обсуждение результатов

Следует отметить, что подход ландшафтного распараллеливания уже успешно применяется для таких задач прогноза лесной пожарной опасности, как определение лесопожарного созревания слоя ЛГМ [9], расчет выбросов вредных веществ и теплоты от множественных очагов лесных пожаров. В данной статье представлено обобщение данного подхода на задачи определения вероятности возникновения и числа лесных пожаров на крупных лесопокрытых территориях.

Кроме того, в перспективе возможно применение данного подхода к задачам расчета распространения лесных пожаров. Необходимо будет только предварительно разработать технологии определения пространственной корреляции лесных выделов и определения вероятности потенциально возможного сценария распространения лесного пожара.

Научно-практическая значимость разработанной методики определения вероятности возникновения лесных пожаров велика, так как впервые, в отличие от методик, разработанных в США, Канаде, Южной Европе и России, разработан вероятностный критерий лесной пожарной опасности, который учитывает не только метеоданные, но и грозовую активность, уровень антропогенной нагрузки. Следует заметить, что год от года вследствие усиления человеческой деятельности (т.е. уровня антропогенизации) значимость учета антропогенной нагрузки будет только возрастать. Разработанная методика реализована в виде параллельного программного комплекса.

Подход ландшафтного распараллеливания является проблемно-ориентированным, но в пределах области своего приложения является достаточно гибким и универсальным. Разработаны прототип аппаратно-

программного комплекса для лесопожарного мониторинга крупных лесопокрытых территорий и параллельный программный комплекс прогноза возникновения лесных пожаров на крупных лесопокрытых территориях.

Разработанный аппаратно-программный комплекс (вычислительный кластер, параллельный программный комплекс, сопутствующее методическое обеспечение) используется в учебном и научно-исследовательском процессах на кафедре физической и вычислительной механики ММФ ТГУ.

Заключение

Таким образом, в данной работе представлен детерминированно-вероятностный подход к определению вероятности возникновения и числа лесных пожаров. Разработан новый проблемно-ориентированный подход ландшафтного распараллеливания. Как показал анализ результатов проведенного исследования, данный подход очень эффективен при решении задач лесопожарного созревания слоя ЛГМ, определении вероятности возникновения лесных пожаров, оценке экологических последствий лесных пожаров. Параллельная программа достигает практически максимально возможного ускорения, когда ускорение практически пропорционально числу задействованных процессоров.

Детерминированно-вероятностный подход определения вероятности возникновения лесных пожаров и подход ландшафтного распараллеливания позволяют создавать очень эффективные параллельные программы для прогноза лесной пожарной опасности на больших лесопокрытых территориях. В дальнейшем планируется апробация рассмотренного подхода к прогнозу лесной пожарной опасности на территории Тимирязевского лесхоза Томской области.

Литература

1. Исаев А.С., Коровин Г.Н., Титов С.П. и др. Экологические проблемы поглощения углекислого газа посредством лесовосстановления и лесоразведения в России: Аналитический обзор. – М.: Центр экологической политики, 1995. – 156 с.
2. Проект организации и развития лесного хозяйства Тимирязевского мехлесхоза Томского лесохозяйственного территориального производственного объединения Министерства лесного хозяйства РСФСР. Т. III: Таксационные описания Тимирязевского лесничества. Кн. 3: Кварталы 91-145. Инв. № 390. – Томск: Гослезхоз СССР. Всесоюзное объединение «Леспроект». Западно-Сибирское лесоустроительное предприятие, 1990. – 400 с.
3. Малышкин В.Э. Основы параллельных вычислений. – Новосибирск: Изд-во НГТУ, 1998. – 60 с.

4. Snir M., Otto S.M., Huss-Lederman S. et al. MPI: The Complete Reference. – Boston: MIT Press, 1996.
5. Вшивков В.А., Краева М.А., Малышкин В.Э. Параллельные реализации метода частиц // Программирование. − 1997. − № 2. − С. 39−51.
6. Барановский Н.В. Влияние антропогенной нагрузки и грозовой активности на вероятность возникновения лесных пожаров // Сибирский экологический журнал. − 2004. − № 6. − С. 835−842.
7. Аветисян А.И., Гайсарян С.С., Самоваров О.И. Возможности оптимального выполнения параллельных программ, содержащих простые и итерированные циклы, на неоднородных параллельных вычислительных системах с распределенной памятью // Программирование. − 2002. − № 1. − С. 38−54.
8. Барановский Н.В., Гришин А.М., Тимченко С.В. Разработка аппаратно-программного комплекса для прогноза лесопожарного созревания лесных горючих материалов // Труды Международной научно-практической конференции МВС-2002. − Таганрог: НИИ МВС ТРТУ, 2002. − С. 236−240.
9. Baranovsky N.V., Grishin A.M. Prediction of forest fire maturity of forest fuel layer using landscape parallelization // Вычислительные технологии. − 2002. − Т. 7, − ч. 1. [Совместный выпуск Вестник КазНУ. − № 4 (32)]. − С. 37−44.

К ВЫБОРУ СПОСОБА ДЕКОМПОЗИЦИИ ПРИ ЧИСЛЕННОМ РЕШЕНИИ СИСТЕМ СВЯЗАННЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ НА МНОГОПРОЦЕССОРНОЙ ТЕХНИКЕ С РАСПРЕДЕЛЕННОЙ ПАМЯТЬЮ

Е.А. Данилкин, А.В. Старченко

Томский государственный университет

Введение

При использовании параллельных компьютеров с локальной памятью первым по значимости является вопрос о возможности проведения эффективных параллельных вычислений для конкретной решаемой задачи. Речь идет о пересмотре взглядов на стандартные алгоритмы, поскольку некоторые из них становятся неприменимыми при увеличении числа процессоров, другие, наоборот, приобретают большое значение. Определяющим фактором для оценки перспективности алгоритма являются затраты на межпроцессорные обмены, которые необходимы для его реализации. Если алгоритм требует большого числа пересылок, то можно не получить ускорения и тем самым не будет достигнута цель применения параллельных компьютеров – уменьшение времени вычислений. Проблему поможет решить удачное распределение данных по процессорам, при котором затраты на пересылки будут минимальны.

Цель данной работы заключалась в исследовании возможных подходов решения систем связанных линейных дифференциальных уравнений в частных производных методом конечного объёма на многопроцессорной системе с распределённой памятью и в выборе оптимального способа декомпозиции с точки зрения минимизации межпроцессорных обменов.

Математическая постановка задачи

Требуется решить систему двух связанных линейных дифференциальных уравнений в единичном квадрате ($(x, y) \in (0,1) \times (0,1)$, G – граница) с граничными условиями третьего рода

$$\begin{cases} \dfrac{\partial}{\partial x}\left(D_1 \dfrac{\partial u}{\partial x} \right) + \dfrac{\partial}{\partial y}\left(D_1 \dfrac{\partial u}{\partial y} \right) + E_1 \dfrac{\partial u}{\partial x} + B_1 v = F_1; \qquad \alpha_1 \left. \dfrac{\partial u}{\partial n} \right|_G + \beta_1 u|_G = \gamma_1; \\ \dfrac{\partial}{\partial x}\left(D_2 \dfrac{\partial v}{\partial x} \right) + \dfrac{\partial}{\partial y}\left(D_2 \dfrac{\partial v}{\partial y} \right) + E_2 \dfrac{\partial v}{\partial x} + B_2 u = F_2; \qquad \alpha_2 \left. \dfrac{\partial v}{\partial n} \right|_G + \beta_2 v|_G = \gamma_2, \end{cases} \tag{1}$$

где D_i, E_i, B_i, F_i $(i = 1,2)$ – достаточно гладкие функции от x и y, а $\alpha_i, \beta_i, \gamma_i$ $(i = 1,2)$ – константы.

Численный метод решения

Аппроксимация дифференциальной задачи осуществлялась на основе метода конечного объема. Основная идея этого метода заключается в разбиении расчетной области на непересекающиеся, граничащие друг с другом конечные объемы так, чтобы один узел расчетной сетки содержался только в своем конечном объеме. Разбив таким образом расчетную область, интегрируем каждое уравнение системы по каждому конечному объему. При вычислении интегралов используются интерполяционные квадратурные формулы численного интегрирования для зависимых величин, входящих в (1). В результате такого приближенного интегрирования получается дискретный аналог системы дифференциальных уравнений:

$$\begin{cases} ap_{ij}u_{ij} = ae_{ij}u_{i+1\,j} + aw_{ij}u_{i-1\,j} + an_{ij}u_{ij+1} + as_{ij}u_{ij-1} + ad_{ij}v_{ij} + b_{ij}; & 0 \le i \le Nx; \\ ap'_{ij}v_{ij} = ae'_{ij}v_{i+1\,j} + aw'_{ij}v_{i-1\,j} + an'_{ij}v_{ij+1} + as'_{ij}v_{ij-1} + ad'_{ij}u_{ij} + b'_{ij}; & 0 \le j \le Ny. \end{cases} \quad (2)$$

Матрица полученной системы сеточных уравнений (2) имеет вид, показанный на рис. 1. Здесь U,V – компоненты вектор-столбца неизвестных; B – столбец свободных членов.

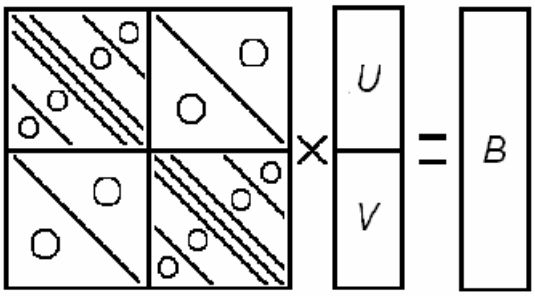


Рис. 1. Представление системы (2) в блочном виде

$$U = \{u_{i\,j}\},\ V = \{v_{i\,j}\},\ B = \begin{pmatrix} \{b_{i\,j}\} \\ \{b'_{i\,j}\} \end{pmatrix}.$$

Матрица системы (обозначим ее через A) имеет большое количество нулевых элементов, а ненулевые выстроены в диагонали, как показано на рис. 1. Матрица A состоит из четырех больших блоков, два из которых пятидиагональные матрицы и два содержат лишь по ненулевой главной диагонали.

Метод решения систем линейных алгебраических уравнений

В качестве метода решения системы сеточных уравнений рассматривается метод BiCGStab [3] с предобусловливанием по Булееву [1, 2]. В настоящее время этот метод является одним из наиболее быстродействующих по времени и количеству итераций. Он относится к семейству методов вариационно-итерационного типа и применим для решения знакоопределенных СЛАУ с несимметричной в общем случае матрицей, для которой неизвестны спектральные свойства.

Практически все современные итерационные алгоритмы решения задач вида $Ax = b$ относятся к методам подпространств Крылова, основанным на минимизации следующего функционала: $F(x) = (Ax, x) - 2(b, x)$. Решение связано с построением системы сопряженных векторов и каждое следующие приближение ищется в направлении нового полученного вектора, ис-

ходя из условия минимума функционала в этом направлении. Ниже представлен алгоритм метода BiCGStab [3].

BiCGStab алгоритм с предобусловливанием:

x_0 – начальное приближение; $r_0 = b - Ax_0$;

$\tilde{r}_0$ – произвольный вектор, для которого $(\tilde{r}_0, r_0) \neq 0$, например, $\tilde{r}_0 = r_0$;

$\rho_0 = \alpha = \omega_0 = 1;\ v_0 = p_0 = 0;$

for $i = 1,\ 2,\ 3,\ \ldots$;

$\rho_i = (\tilde{r}_0, r_{i-1});$

$\beta_i = (\rho_i / \rho_{i-1})(\alpha / \omega_{i-1});$

$p_i = r_{i-1} + \beta(p_{i-1} - \omega_{i-1} v_{i-1});$

находим y из $Ky = p_i$ $(K \approx A)$;

$v_i = Ay;$

$\alpha = \rho_i / (\tilde{r}_0, v_i);$

$s = r_{i-1} - \alpha v_i;$

находим z из $Kz = s;$

$t = Az;$

$\omega_i = (t, s)/(t, t);$

$x_i = x_{i-1} + \alpha p_i + \omega_i s;$

если x_i достаточно точное, то остановка;

$r_i = s - \omega_i t;$

end;

Как указывается в [3], метод BiCGStab можно считать прямым методом, поскольку при точной арифметике он сойдётся к точному решению за конечное число итераций. Это следует из свойства сопряженности векторов построенной системы.

Предобусловливание

Идея предобусловливания заключается в преобразовании исходной задачи (решение системы линейных уравнений) таким образом, чтобы матрица предобусловленной системы имела лучшие спектральные свойства и полученная система решалась более легко. Под этим понимается, что алгоритм требует меньше вычислительных ресурсов за меньшее число итераций.

Для ускорения процесса сходимости и уменьшения числа итераций в данной работе используется процедура предобусловливания по методу неполной факторизации Булеева, формулы которой получаются из формул явного метода Булеева исключением итерационных членов [1,2].

Алгоритм предобусловливания (решение систем вида $Ky = p_i$ или $Kz = s$, $K \approx A$) состоит из двух этапов: прямого и обратного хода. На первом шаге вычисляются прогоночные коэффициенты, формулы для определения которых имеют рекуррентный вид (где Θ – параметр компенсации $0 \le \Theta < 1$):

$$\begin{cases} P_{ij} = \dfrac{ae_{ij}}{g_{ij}},\ Q_{ij} = \dfrac{an_{ij}}{g_{ij}},\ T_{ij} = \dfrac{(b_{ij} + aw_{ij}T_{i-1\,j} + as_{ij}T_{ij-1})}{g_{ij}}; & i = 1,2,...,Nx-1; \\ g_{ij} = ap_{ij} - aw_{ij}(P_{i-1\,j} + \Theta Q_{i-1\,j}) - as_{ij}(Q_{ij-1} + \Theta P_{ij-1}); & j = 1,2,...,Ny-1. \end{cases} \quad (3)$$

А затем решение $\{y_{i\,j}\}$ находится по формуле

$$y_{ij} = P_{ij}y_{i+1\,j} + Q_{ij}y_{ij+1} + T_{ij};\quad i = Nx-1,...,0;\ \ j = Ny-1,...,0. \quad (4)$$

Отметим, что использующееся в настоящее время предобусловливание по методу Холесского или LU-факторизации для задач вида (1) менее эффективно, чем предобусловливание по Булееву [2].

Параллельная реализация

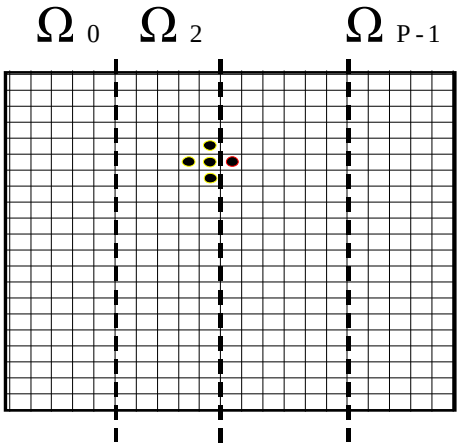


Рис. 2. Декомпозиция расчетной области

Распараллеливание метода BiCG-Stab для случая решения одного дифференциального уравнения осуществляется следующим образом. Используется одномерная декомпозиция сеточной области по индексу j, то есть каждому процессорному элементу выделяется определенное количество узлов сетки Ω, для которых он будет производить вычисления.

При аппроксимации дифференциальной задачи (1) использовался шаблон «крест», поэтому для вычисления очередного приближения в приграничных узлах подобласти Ω_k ($0 \le k \le P-1$, где P – число процессоров) необходимо пересылать значения с соседнего процессора.

Внутренний цикл алгоритма BiCGStab содержит два матрично-векторных произведения, четыре скалярных произведения, шесть операций saxpy и небольшое количество скалярных операций. Все эти алгебраические операции легко распараллеливаются [4].

Структура матрицы A (для случая одного уравнения) позволяет организовать вычисления так, что обмены требуются лишь для пересылки приграничных значений сеточной функции и сбора и рассылки частичных сумм скалярных произведений.

При предобусловливании системы линейных уравнений допустимо применять асинхронный вариант метода Булеева, когда при вычислении значений прогоночных коэффициентов в (3) и нахождении решения в (4) в каждой подобласти декомпозиции на ее границе используются значения сеточной функции $\{y_{ij}\}$ с предыдущей итерации.

Для случая решения системы сеточных уравнений с матрицей, представленной на рис. 1, ситуация меняется, и неудачное распределение данных по процессорным элементам может привести к значительному возрастанию пересылок, что в свою очередь сведёт на нет ускорение параллельной программы.

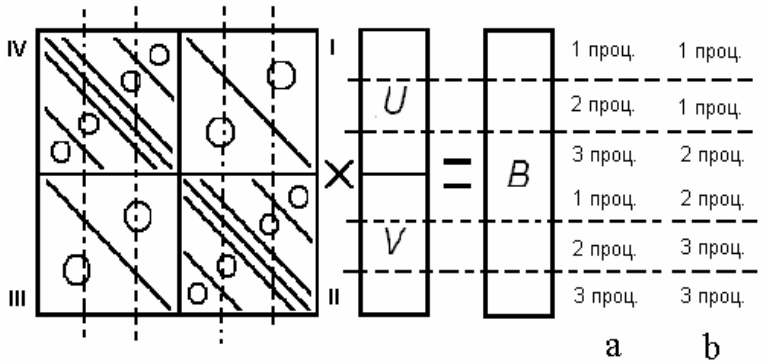


Рис. 3. Два варианта декомпозиции данных по процессорам (a–декомпозиция, b–декомпозиция)

На рис. 3 столбец a представляет оптимальный вариант декомпозиции данных, позволяющий избежать использования дополнительных пересылок. Дело в том, что при численном решении системы дифференциальных уравнений (1) связи накладываются не только на значения сеточной функции в точках шаблона u_{ij} и $u_{i\pm1\,j\pm1}$, но и на значения неизвестных, принадлежащих одной подобласти Ω_k (u_{ij} и v_{ij}).

Идея предложенного способа декомпозиции заключается в следующем. По процессорным элементам последовательно распределяются не неизвестные u_{ij}, v_{ij}, а каждому процессорному элементу выделяется целая сеточная подобласть Ω_k со всеми неизвестными, принадлежащими ей. Таким образом, все связанные значения сеточной функции распределены одному процессорному элементу.

При другом способе декомпозиции (рис. 3, столбец b) потребуется больше времени на межпроцессорную пересылку данных, как, например,

при размещении на одном процессорном элементе только значений $\{u_{ij}\}$ или только $\{v_{ij}\}$.

Были получены теоретические оценки ускорения расчетного метода для различных способов декомпозиции данных.

$$S_p^{(1)} = \frac{P}{1+\frac{P\cdot 24}{61\cdot\sqrt{N}}\cdot\chi}; \qquad S_p^{(2)} = \frac{P}{1+\frac{P\cdot 24}{61\cdot\sqrt{N}}\cdot\chi+\frac{\chi}{61}}.$$

Здесь $S_p^{(1)}$ – ускорение алгоритма при предложенном способе распределения данных по процессорным элементам (рис. 3, столбец a); $S_p^{(2)}$ – теоретическое ускорение для случая, когда возникает необходимость обменов большими частями решения. Например, при декомпозиции, представленной на рис. 3 (столбец b), первому процессору для проведения расчетов потребуются значения неизвестных $\{v_{ij}\}$ со 2-го и 3-го процессоров. Здесь P – число процессоров; N – размерность задачи; χ – отношение времени, затрачиваемого на пересылку одного числа, ко времени одной арифметической операции с плавающей точкой.

Для подтверждения полученных теоретических оценок была выбрана следующая тестовая задача:

$$\begin{cases} \frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+\frac{\partial u}{\partial x}+v=1; & u\big|_G=2; \\ \frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}+\frac{\partial v}{\partial x}+u=2; & v\big|_G=1. \end{cases} \qquad (5)$$

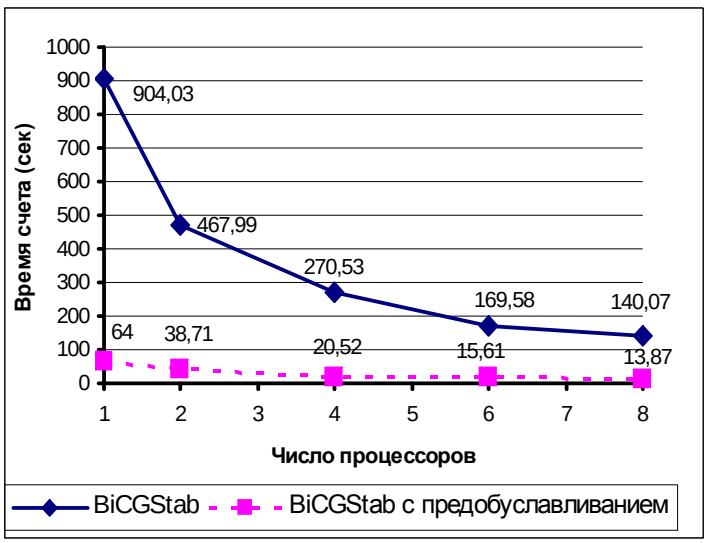


Рис. 4. Зависимость времени счета от числа процессоров

В качестве начального использовалось нулевое приближение. Приближенное решение соответствующих сеточных систем считалось достигнутым, если норма невязки и норма ошибки становились меньше $\varepsilon = 10^{-5}$.

Были проведены расчеты на кластере ИОА СО РАН для сетки 480x480 (http://www.iao.ru/ru/resources/equip/cluster/). На рис. 4 представлен график зависимости времени вычисления от числа используемых процессорных элементов. Результаты численного эксперимента показали, что использование предобусловливателя более чем на порядок сокращает временные затраты на решение системы линейных алгебраических уравнений.

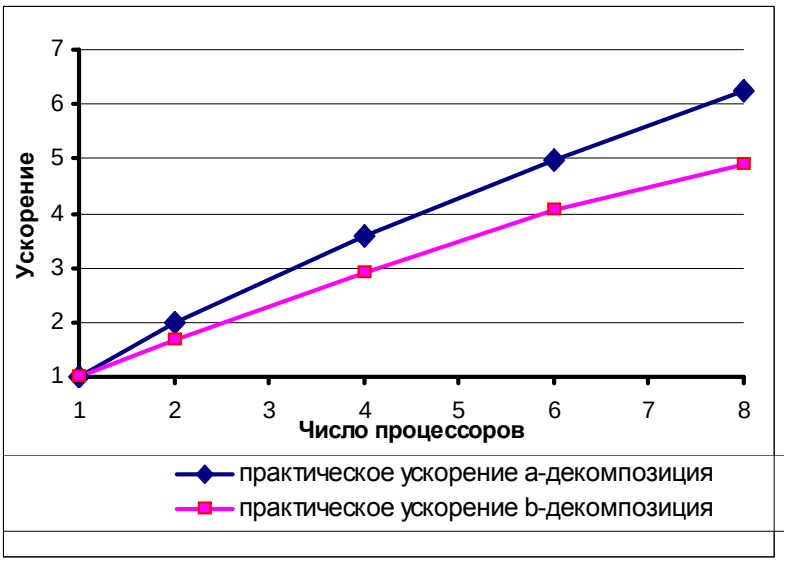


Рис. 5. Ускорение метода для рассматриваемых способов декомпозиции данных

На рис. 5 представлено ускорение для разных способов распределения данных по процессорным элементам. Видно, что при использовании предложенного способа декомпозиции ускорение на единицу больше, чем при альтернативном способе распределения данных, а это даст существенный выигрыш во времени при решении больших задач.

Литература

1. Ильин В.П.. Методы неполной факторизации для решения алгебраических систем. – М.: Наука, 1995. – 287 с.
2. Старченко А.В. // Вестник Томского государственного университета. − 2003. − №10. − С. 70−80.
3. Van der Vorst H.A. Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetrical linear systems // SIAM Journal. − 1992. − Vol. 13, № 2. − P. 631−644.
4. Ортега Дж. Введение в параллельные и векторные методы решения линейных систем. – М.: Мир, 1991. – 364 с.

ПРОБЛЕМЫ ОРГАНИЗАЦИИ МНОГОУРОВНЕВОЙ ПАМЯТИ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

М.С. Сущенко, С.П. Сущенко

Томский государственный университет

Введение

Разрыв скорости обработки данных в центральном процессоре и скорости доступа к адресуемым операндам в оперативной памяти современных компьютеров составляет 1-2 порядка. Очевидно, что высокое быстродействие вычислительной системы в значительной мере определяется организацией ее многоуровневой памяти. Главными требованиями, которым должна удовлетворять подсистема памяти, являются достаточно большая емкость, высокое быстродействие и экономическая эффективность с точки зрения технической реализации [1]. Желательно также, чтобы при минимальных физических размерах память обладала как можно большей информационной емкостью. Удовлетворить все эти требования одновременно в одном устройстве невозможно, поэтому обычно комбинируют несколько запоминающих устройств с различными параметрами, добиваясь создания комплексного решения с требуемыми характеристиками. Многоуровневая организация памяти позволяет сгладить разрыв между скоростью обработки данных центральным процессором и скоростью доступа к адресуемым объектам в оперативной памяти вычислителя. Эффективность доступа к иерархической памяти определяется не только быстродействием и емкостью отдельных уровней подсистемы памяти, но и набором архитектурных параметров, важнейшими среди которых являются коэффициент ассоциативности и глубина неблокируемости кэш-памяти. Ассоциативность определяет способ отображения пространства адресов оперативной памяти в кэш-память и, наряду со стратегией вытеснения кэш-строк при конфликте адресов, в значительной мере влияет на степень локализации прикладных задач в памяти верхнего уровня. Степень локализации в свою очередь определяет вероятность «попадания» адресуемого объекта в кэш и время доступа к нему. Как правило, различные уровни иерархической памяти работают асинхронно, поэтому между ними предусмотрены буферные устройства, которые хранят адреса востребованных вычислителем элементов данных и позволяют параллельно выполнять операции доступа к различным уровням. Емкость межуровневого буферного устройства определяет параметр глубины неблокируемости кэша и задает потенциально достижимую степень параллелизма выполнения совокупности транзакций доступа к различным уровням иерархической памяти.

В основе работы кэш-памяти лежит принцип временной и пространственной локальности программ [1,2]. Согласно принципу временной локальности, обращения к памяти носят не случайный характер, а выполняются в соответствии с исполняемой программой. Тогда при считывании данных из памяти с высокой степенью вероятности можно предположить, что в ближайшем будущем программа опять обратится к этим данным, и их целесообразно хранить в течение некоторого времени. В соответствии с принципом пространственной локальности весьма вероятно то, что в ближайшем будущем программа обратится к ячейке, которая следует за той, к которой она обращается в текущий момент времени. Принцип пространственной локальности предполагает считывание в кэш нескольких соседних ячеек памяти (блока информации). Каждый блок хранится в строке буфера, а набор таких строк составляет кэш-память. Таким образом, в пределах небольших временных интервалов обращения к основной памяти за командами и данными сосредоточиваются в ограниченной области адресного пространства.

Проблемы организации подсистемы памяти многопроцессорных систем

При построении вычислительных систем с массовым параллелизмом и вычислительных кластеров в качестве отдельных узлов широко применяются многопроцессорные вычислители с общей памятью (SMP архитектурой). Одной из основных проблем масштабируемых многопроцессорных вычислительных систем с SMP архитектурой является построение распределенной разделяемой памяти [4−6], имеющей единое адресное пространство и доступ к объектам через операции чтения и записи. Важнейшая задача при этом состоит в том, как извещать процессоры об изменениях, вызванных выполнением команд записи. Необходимость увеличения быстродействия вычислителей приводит к применению многоуровневой памяти и механизмов отложенной модификации удаленных копий одного и того же экземпляра данных, находящегося в кэшах различных процессоров и оперативной памяти. При реализации изменений удаленных копий применяется коммуникационный протокол, обеспечивающий доставку изменений, и протокол согласованности или когерентности состояния памяти, предотвращающий использование неизмененных копий модифицированного в одном из процессоров элемента данных. Для сохранения когерентности данных в соответствии с моделью строгой состоятельности памяти операция чтения из разделяемой памяти должна возвращать последнее записанное значение [4,6]. Поэтому при модификации одной из множества копий остальные должны быть либо изменены, либо объявлены несостоятельными (не соответствующими последнему изменению). Выполнение этих действий требует реализации средств быстрого распространения изменений,

однако в любом случае следование модели строгой состоятельности (немедленного изменения значения элемента данных на всех уровнях иерархической памяти) ведет к снижению производительности. Для повышения производительности применяют модели ослабленной состоятельности памяти, допускающие появление несогласованных копий данных в ходе их параллельной обработки с последующим обеспечением когерентности копий, приводящим к тому же результату исполнения приложения, что и при строгой состоятельности.

Современные микропроцессоры, применяемые для построения многопроцессорных вычислительных систем, имеют встроенные аппаратные средства обеспечения когерентности данных в иерархической памяти [5]. Когерентность кэшей процессоров вычислителя обеспечивается с помощью межузловых (межмодульных) пересылок. Размер разделяемой процессорами единицы данных обычно составляет кэш-строку. Методы реализации когерентности определяются видом памяти вычислительной системы (сосредоточенная или распределенная) и типом коммуникационной среды, объединяющей память и процессоры. Аппаратная реализация когерентности универсальна и не требует наличия в программах дополнительного кода для организации разделяемой памяти. При этом обеспечивается минимальная задержка согласования копий данных и высокая пропускная способность всего механизма когерентности.

Для вычислительных систем с сосредоточенной памятью, реализуемых, как правило, в симметричных многопроцессорных архитектурах, наиболее широкое распространение получил алгоритм поддержки когерентности кэшей MESI (Modified, Exclusive, Shared, Invalid) [7]. Этот алгоритм обепечивает когерентность кэш-памяти с обратной записью и минимизирует пересылки данных между основной памятью и кэшем [3]. В общем случае в SMP-системах возможны как модификация данных, так и объявление модифицируемых данных несостоятельными.

Алгоритм MESI предполагает, что вычислитель имеет общую разделяемую память, в каждом процессоре содержится локальная кэш-память, все процессоры объединены между собой и с основной памятью посредством шины. Кроме того, к шине подключены внешние устройства. Ключевым в данной конструкции является то, что все действия, выполняемые на шине по инициативе процессоров и внешних устройств с копиями строк в кэшах процессоров и основной памяти, доступны для отслеживания всем процессорам. При этом в каждый момент времени по шине информация передается одним источником, а воспринимают ее все подключенные к шине абоненты. Если в качестве коммуникационной среды вместо шины используется другой тип соединения, то для работоспособности алгоритма MESI необходимо обеспечение указанного «прослушивания» транзакций всеми процессорами вычислительной системы.

Вычислительные системы с памятью, физически распределенной по вычислительным узлам (модулям), но имеющей единое пространство адресов (единую карту памяти), применяют различные алгоритмы реализации когерентности. Простейший из них заключается в том, что при каждом промахе в кэш в любом процессоре инициируется запрос требуемой строки из резидентного блока памяти (того блока памяти, в котором эта строка размещена). Запрос передается через коммутатор или другую коммуникационную систему в узел с резидентным для строки блоком памяти, откуда необходимая строка пересылается в узел, в котором произошел промах. В соответствии с такой процедурой обеспечивается, в том числе, начальное заполнение кэшей. Каждый узел при этом для каждой резидентной строки ведет список узлов, в кэшах которых размещается данная разделяемая узлами вычислителя строка или организуется распределенный по узлам список этих строк. Более совершенным является алгоритм DASH [3]. Явная реализация когерентности предусматривает применение программистом специальных команд для работы с локальной памятью и управления контроллерами каналов коммуникационной среды, связывающей узлы вычислительной системы с массовым параллелизмом. Основная задача разработчика приложения при этом – эффективное программирование параллельных процессов, совмещающих вычисления и передачу данных между узлами и минимизирующих объем передаваемой информации.

Применение явной когерентности в создаваемых вычислительных системах обусловлено либо слишком большим временем, либо недопустимо высокими аппаратными затратами на реализацию неявного механизма когерентности. Причина этого состоит в высокой сложности применения внутреннего механизма когерентности иерархической памяти узла по сравнению с использованием предусмотренного для работы с внешними устройствами механизма когерентности. Как правило, элементами разделяемой памяти при программной реализации когерентности являются страницы, что обусловлено использованием аппаратных средств организации виртуальной памяти для обнаружения операций записи в разделяемые страницы. Однако относительно большой размер разделяемых страниц создает проблему ложного разделения, при котором конфликтными признаются модификации разными процессорами различных ячеек памяти одной страницы, не конфликтующие между собой [3,6].

Анализ моделей многоуровневой памяти

Основной целью исследования подсистемы памяти является обнаружение функциональных и численных зависимостей между параметрами памяти различных уровней и операционными характеристиками вычислительной системы. При этом необходимо выделить самые существенные факторы, определяющие данные зависимости, так как преувеличенно точ-

ное детальное описание процессов функционирования вычислителя часто мало способствует достижению цели (получению общих закономерностей), а чрезмерно необоснованное упрощение слишком идеализирует реальные процессы. Среди важнейших факторов следует выделить количество уровней иерархии, емкость, ассоциативность и глубину неблокируемости кэш-памяти различных уровней, размер блока и стратегию вытеснения кэш-строк при конфликте адресов, распределение прикладных задач в оперативной памяти, модель состоятельности, реализованную в многопроцессорной вычислительной системе для обеспечения когерентности многоуровневой памяти [2,8−11]. Основными показателями эффективности подсистемы памяти являются вероятность попадания в кэш, среднее время доступа к адресуемым объектам и пропускная способность. Для понимания узких мест и механизма влияния основных факторов на быстродействие вычислительной системы наиболее эффективным инструментом является аналитическое моделирование вычислительных процессов, позволяющее наглядно увидеть достоинства и недостатки различных архитектур [8]. Аналитические модели обычно представляют собой набор уравнений, связывающих показатели производительности с параметрами системы. Однако формализация вычислительных процессов (особенно допускающая аналитическое решение) часто предусматривает их идеализацию, после чего возникают вопрос адекватности аналитической модели реальной системе и проблема интерпретации полученных результатов. Поэтому наряду с аналитическими методами широко применяются имитационное моделирование и методы тестирования вычислительных систем [16], позволяющие исследовать трудноформализуемые при аналитическом моделировании элементы вычислительного процесса. Кроме того, методы тестирования часто используются для выявления уровня локализации тех или иных приложений в вычислительных системах с заданной архитектурой подсистемы памяти.

Известные исследования быстродействия подсистемы памяти основаны на модели двухуровневой памяти (кэш – оперативная память) [6,8−12,16]. В рамках такой модели при заданном типе распределения прикладных задач в оперативной памяти вычислительной системы обычно анализируются вероятность попадания в кэш и среднее время доступа к иерархической памяти, а также скорость обновления содержимого кэша при переключениях вычислителя на выполнение нового приложения (скорость движения к стационарному состоянию). Анализ факторов, определяющих долю обращений за блоками информации, удовлетворяемых на уровне кэша, выполнен в [8,17−21]. На основе предлагаемых математических моделей, как правило, находятся аналитические и численные оценки операционных показателей для одно- и многопроцессорных вычислительных систем. По результатам исследований отмечается, что основными причинами снижения вероятности попадания в кэш многопроцессорных

вычислителей являются переключения задач на процессорах и снижение полезного объема кэша из-за хранения несостоятельных копий блоков данных в кэшах различных процессоров. В работах [8,9,12,22,23] получены асимптотические соотношения и численные зависимости операционных показателей эффективности подсистемы памяти от объема кэш-памяти, коэффициента ассоциативности, размера блока данных. Из проведенного анализа следует, что рост ассоциативности позволяет снизить отрицательный эффект от явления конфликта адресов одинаково востребованных вычислителем блоков памяти, отображаемых на одну группу кэша. Характерная зависимость вероятности попадания в кэш от коэффициента ассоциативности определяется монотонно возрастающей функцией, а от объема кэш-памяти имеет вид кривой с насыщением. Однако общий аналитический вид операционных характеристик от данных параметров не получен.

Поскольку в большинстве современных вычислительных систем используются архитектуры памяти с тремя и более уровнями иерархии, то, очевидно, следует необходимость разработки моделей многоуровневой подсистемы памяти (с числом уровней больше двух) и уточнения моделей двухуровневой памяти. Кроме того, важной является задача построения подсистемы памяти с оптимальным в смысле некоторого критерия числом уровней иерархии при заданных параметрах каждого уровня (длительностях обращения к памяти, вероятностях промаха, глубинах неблокируемости, ассоциативности, емкости). Оценивание эффективности многоуровневых систем памяти в однопроцессорных вычислителях проводилось в работах [24−26]. В них решается задача минимизации среднего времени обращения к иерархической памяти при известной связи времени доступа к памяти каждого уровня с ее удельной стоимостью и частоты обращения к памяти каждого уровня с ее объемом, а также заданной конфигурации подсистемы памяти (число уровней, вероятности обращения к памяти каждого уровня) и ограничениях на ее общую стоимость. Однако проблема обоснованного выбора рационального числа уровней иерархии не рассматривалась.

Влияние стратегий вытеснения кэш-строк при конфликте адресов обсуждается в [9,11−14,23,27,28]. Отмечается, что наилучшей среди реализуемых стратегией является стратегия LRU (Least Recently Used), которая в то же время характеризуется высокой сложностью и трудоемкостью. На примерах показано, что вероятность попадания в кэш с ростом ассоциативности увеличивается, а от длины блока – имеет унимодальный характер. Наличие оптимального размера блока обусловлено двумя факторами. С одной стороны, размер блока следует увеличивать, чтобы усилить эффект пространственной локализации программного кода и обрабатываемых данных. С другой стороны, рост длины блока приводит к сокращению числа блоков при фиксированной емкости кэша и, как следствие, увеличе-

нию частоты конфликта адресов и, соответственно, интенсивности замещения блоков.

В работах [29−31] отмечается положительное воздействие на операционные характеристики подсистемы памяти увеличения емкости межуровневого интерфейса (параметра глубины неблокируемости). Так, например, в процессоре Pentium 4 отсутствие блокировок кэша обеспечивается наличием 48 и 24 межуровневых буферов для команд чтения и записи соответственно [32]. Однако строгого математического моделирования и обоснования механизма влияния емкости межуровневого буфера на быстродействие иерархической памяти в известных автору исследованиях не проводилось. Из этого следует необходимость построения моделей влияния параметра глубины неблокируемости на вероятностно-временные характеристики подсистемы памяти. Одним из важнейших параметров кэш-памяти, определяющих операционные характеристики подсистемы памяти, является его объем. В целом рост емкости кэша приводит к улучшению вероятностно-временных показателей функционирования иерархической памяти, однако вместе с этим значительно возрастает стоимость вычислительной системы. Тесно связанной с вопросом оптимизации емкости кэш-памяти и общей архитектуры подсистемы памяти является задача определения влияния распределения приложений в оперативной памяти на степень их локализации в кэше и среднее время доступа к адресуемым объектам. Очевидно, что знание этих распределений для различных классов прикладных задач позволит сделать обоснованный выбор архитектуры и параметров подсистемы памяти вычислителя, в том числе емкостей кэш-памяти различных уровней. Однако исследований на эту тему обнаружить не удалось, что обусловливает практическую потребность в анализе взаимосвязи типов решаемых вычислителем задач и соответствующих им рациональных значений параметров подсистемы памяти.

Относительно широко в публикациях представлены исследования эффективности многоуровневой памяти многопроцессорных вычислительных систем [8,10,12,22,27,28]. Основной решаемой здесь задачей является анализ эффективности совместного использования множеством процессоров SMP-вычислителя разделяемой оперативной памяти. Авторы [33,34] предложили математическую модель распределенной иерархической памяти многопроцессорной вычислительной системы, на основе которой получили численные зависимости для среднего времени ожидания запросов на доступ к удаленной памяти на фоне обслуживания более приоритетных запросов к локальной памяти от количества процессоров в системе. Однако при анализе не учитывается параметр глубины неблокируемости кэша. Не рассмотренными также остаются проблемы взаимного влияния на результирующее быстродействие вычислителя существенно неоднородного потока транзакций доступа к иерархической памяти от различных процессоров.

Другой важной проблемой многопроцессорных вычислительных систем является проблема когерентности (согласованности состояния памяти различных уровней), решение которой основано на различных моделях состоятельности памяти [4,6,28]. Работа [4] посвящена классификации моделей состоятельности распределенной памяти и глубокому анализу преимуществ и недостатков каждой из них, а также обсуждению вариантов реализации разделяемой памяти. В [6,28] предложена классификация методов обеспечения когерентности данных в иерархической памяти многопроцессорных вычислительных систем и проведен сравнительный анализ различных методов по критерию эффективного времени обращения процессора к подсистеме памяти. По результатам исследований определены области предпочтения рассмотренных способов реализации когерентности разделяемой различными процессорами информации.

Работы [35−39] посвящены исследованию влияния структурных факторов векторных, конвейерных и параллельных вычислительных систем на их быстродействие. Получены оценки и границы скорости выполнения вычислений. В целом процесс доступа к объектам, хранимым в многоуровневой памяти, хорошо описывается конвейером с числом фаз, равным количеству уровней иерархии, нарастающим временем обработки в каждой последующей фазе и случайным числом последовательных фаз обработки каждой транзакции доступа к подсистеме памяти. Степень загрузки конвейера при этом целиком определяется параметрами глубины неблокируемости и вероятности промаха на каждом уровне. Наибольшую сложность моделирования конвейера с такими свойствами представляет проблема поиска распределений времени пребывания запроса к подсистеме памяти в различных фазах.

Заключение

При анализе проблем организации эффективной подсистемы памяти вычислительных систем выясняется необходимость совершенствования существующих моделей иерархической памяти и разработки на их основе методов расчета архитектурных параметров многоуровневой памяти. Проведенный аналитический обзор результатов, достигнутых в изучении воздействия архитектурных параметров подсистемы памяти и распределения в оперативной памяти выполняемой вычислителем смеси прикладных задач на скорость доступа к хранимым данным, позволил выявить степень проработки вопроса организации быстродействующей иерархической памяти, сформулировать нерешенные задачи и наметить основные направления исследований.

1. Существенным недостатком известных моделей иерархической памяти является отсутствие явных аналитических зависимостей операционных характеристик подсистемы памяти от коэффициента ассоциативности

и стратегии вытеснения срок кэша. В связи с этим необходимы разработка моделей влияния ассоциативности на быстродействие подсистемы памяти и исследование влияния распределения прикладных задач на вероятность попадания адресуемых объектов в кэш.

2. Существующие подходы к анализу подсистемы памяти не учитывают фактор, определяемый конвейерной природой обработки транзакций доступа к иерархической памяти. Степень параллелизма доступа к адресуемым объектам, хранящимся на различных уровнях памяти, определяется параметром глубины неблокируемости кэша. Поэтому представляется актуальным построение моделей влияния данного параметра на вероятностно-временные показатели функционирования вычислителя.

3. Кроме того, в известных исследованиях отсутствуют модели влияния стратегий сохранения целостности данных, модели подсистем памяти с числом уровней больше двух и решение задачи оптимизации числа уровней иерархической памяти, модели влияния числа процессоров на вероятностно-временные характеристики.

Литература

1. Кохонен Т. Ассоциативные запоминающие устройства. – М.: Мир, 1982. – 384 с.
2. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – СПб.: БХВ-Петербург, 2002. – 608 с.
3. Корнеев В.В. Параллельные вычислительные системы. – М.: Нолидж, 1999. – 320 с.
4. Корнеев В.В. Архитектуры с распределенной разделяемой памятью // Открытые системы. − 2001. − № 3. − С. 15−23.
5. Корнеев В.В. Будущее высокопроизводительных вычислительных систем // Открытые системы. − 2003. − № 5. − С. 10−17.
6. Adve S.V., Charachorloo K. Shared memory consistency models: A Tutorial // IEEE Comput. Magn. − 1999. − Vol. 29, № 12. − P. 66−76.
7. Корнеев В.В., Киселев А.В. Современные микропроцессоры. – М.: Нолидж, 1998. – 240 с.
8. Богуславский Л.Б., Ляхов А.И. Методы оценки производительности многопроцессорных систем. – М.: Наука, 1992. – 126 с.
9. Лускинд Ю.И. Буферные запоминающие устройства типа кэш // Зарубежная радиоэлектроника. − 1990. − № 4. − С. 29−42.
10. Ляхов А.И. Многопараметрический анализ эффективности использования кэш-памяти на основе асимптотических методов // Автоматика и телемеханика. − 1989. − № 11. − С. 155−165.
11. Танненбаум Э. Архитектура компьютера. – СПб.: Питер, 2002. – 704 с.

12. Феррари Д. Оценка производительности вычислительных систем. – М.: Мир, 1981. – 576 с.
13. McKee S.A., Klenke R.H., Wright K.L. et al. Smarter memory: Improving bandwdth for streamed references // IEEE Comput. Magn. − 1998. − Vol. 31, № 7. − P. 54−63.
14. Smith A. Cache memories // ACM Computing Surveys. − 1982. − Vol. 14, № 3. − P. 473−530.
15. Старченко А.В., Есаулов А.О. Параллельные вычисления на многопроцессорных вычислительных системах. – Томск: Изд-во Том. ун-та, 2002. – 56 с.
16. Французов Д. Оценка производительности суперкомпьютеров // Открытые системы. − 1995. − № 6. − С. 48−51.
17. Mattson R.L. Evaluation of multilevel memories // IEEE Trans. Magn. − 1971. − Vol. MAG-7, № 12. − P. 814−819.
18. Patel J.H. Analysis of multiprocessors with private cache memories // IEEE Trans. Comput. − 1982. − Vol. 31, № 4. − P. 296−304.
19. Patel J.H. Performance of processor-memory interconnections for multiprocessors // IEEE Trans. on Comput. − 1981. − № 11. − P. 771−780.
20. Smith J.E., Sohi G.S. The microarchitekture of superscalar processors // Proc. of IEEE. − 1995. − Vol. 83, № 12. − P. 1609−1622.
21. Smith A. Cache evaluation and the impact of work-load choice // SIGARCH Newslett. − 1985. − Vol. 13, №. 3. − P. 64−73.
22. Богуславский Л.Б., Ляхов А.И., Шевчик К.С. Сравнительный анализ стратегий доступа с обратной связью в многопроцессорных системах // Автоматика и телемеханика. − 1996. − № 5. − С. 160−176.
23. Смирнов Р.В. Анализ функционирования двухуровневой иерархической памяти процессора // Вопросы радиоэлектроники. Сер. ЭВТ. − 1980. − Вып. 5. − С. 3−11.
24. Юрченко А.С. К выбору конфигурации иерархической системы памяти // Автоматика и телемеханика. − 1985. − № 4. − С. 137−139.
25. Rege S.L. Cost, performance and size tradeoffs for different levels in a memory hierarchy // Computer. − 1976. − Vol. 19. − P. 43−51.
26. Welch T.A. Memory hierarchy configuration analysis // IEEE Trans. Comput. − 1978. − Vol. 27, № 5. − P. 408−413.
27. Башарин Г.П., Богуславский Л.Б., Штейнберг В.И. Анализ конфликтов в общей памяти мультипроцессорных систем // Автоматика и вычислительная техника. − 1980. − № 6. − С. 27−32.
28. Брехов О.М., Слуцкин А.И. Исследование способов обеспечения идентичности информации в буферной памяти многопроцессорных вычислительных систем // Автоматика и телемеханика. − 1988. − № 8. − С. 156−169.

29. Кузьминский М. Архитектура серверов HP Superdome // Открытые системы. − 1997. − № 3. − С. 9−13.
30. Кузьминский М. Краткий обзор IA-64 // Открытые системы. − 1999. − № 9−10. − С. 8−14.
31. Кузьминский М. Микроархитектура Itanium // Открытые Системы. − 2001. − № 9. − С. 8−14.
32. Carmean D. Inside the Pentium 4 Processor Microarchitecture // Intel Development Forum. − 2000.
33. Корячко В.П., Сускин В.В. Анализ эффективности распределения данных в памяти специализированных микропроцессорных вычислительных систем // Автоматика и вычислительная техника. − 1984. − № 2. − С. 66−68.
34. Корячко В.П., Сускин В.В. Анализ функционирования микропроцессорной вычислительной системы с буферной памятью // Автоматика и вычислительная техника. − 1985. − № 3. − С. 75−77.
35. Головкин Б.А. Анализ факторов, влияющих на скорость вычисления в параллельных и конвейерных системах // Автоматика и телемеханика. − 1988. − № 5. − С. 152−164.
36. Головкин Б.А. Вычислительные системы с большим числом процессоров. – М.: Радио и связь, 1995. – 320 с.
37. Головкин Б.А. Параллельные вычислительные системы. – М.: Наука. Гл. ред. физ.-мат. лит-ры, 1980. – 519 с.
38. Головкин Б.А. Соотношения между показателями производительности супер-ЭВМ и метод ее оценивания // Автоматика и телемеханика. − 1989. − № 12. − С. 141−152.
39. Коган Я.А., Ляхов А.И., Нерсесян С.Г. Асимптотический анализ эффективности использования кэш-памяти в многопроцессорных системах // Автоматика и телемеханика. − 1986. − № 11. − С. 142−151.

ИСПОЛЬЗОВАНИЕ АСИНХРОННО ПОСТУПАЮЩИХ ДАННЫХ В ПОТОКОВОЙ МОДЕЛИ ВЫЧИСЛЕНИЙ*

А.И. Легалов

КГТУ, Красноярск

Введение

Асинхронное управление широко используется в параллельном программировании. На его основе разработаны различные модели вычислений. Одной из основополагающих работ является книга Хоара [1], в которой описаны базовые методы асинхронного взаимодействия и подведен соответствующий теоретический фундамент. В схемах потока данных (СПД) тоже используется асинхронное управление, которое связывается с моментами готовности операндов той или иной выполняемой операции [2]. Это позволяет говорить о том, что данный подход к вычислениям в потенциале обеспечивает достижение максимального параллелизма, зависящего только от информационных связей решаемой задачи.

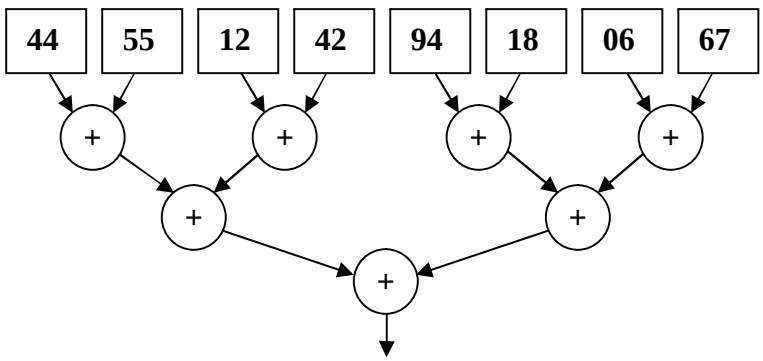


Рис. 1. Суммирование с использованием древовидной свертки

Вместе с тем следует отметить, что известные на данный момент СПД не позволяют эффективно описывать более гибкие методы обработки асинхронных потоков данных, что можно показать на примере нахождения суммы элементов вектора. Большинство потоковых языков используют в вычислениях заранее подготовленный исходный массив. При наличии в этом массиве всех элементов параллельные вычисления можно организовать на основе древовидной свертки, когда в начале суммируются пары из исходного массива, затем результаты, полученные после суммирования этих пар и т.д. до получения окончательного результата. Схема такого

* Работа выполнена при поддержке РФФИ, грант № 05-07-90201.

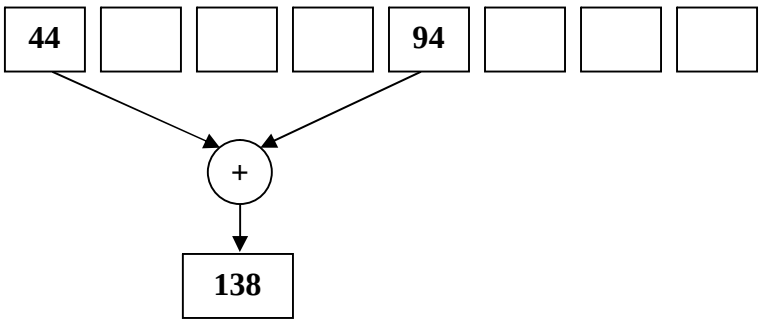


Рис. 2. Возможность суммирования при наличии двух любых элементов

суммирования для массива, состоящего из восьми элементов, представлена на рис. 1.

Следует отметить, что существуют задачи, в которых обрабатываемый массив заранее не известен, а формируется уже в ходе выполнения вычислений. Например, к моменту вызова функции суммирования возможно наличие только 1-го и 5-го элементов массива (рис. 2).

Специфика задачи позволяет не дожидаться всех данных, осуществляя суммирование любых появившихся элементов. Если к окончанию операции суммирования, представленной на рис. 2, в массив поступил только один новый элемент, то его можно сложить с только что вычисленным результатом (рис. 3) до появления других чисел и т.д.

Таким образом, можно оперативно проводить обработку любых поступивших элементов и сформированных промежуточных результатов. Однако подобный метод управления вычислениями в существующих потоковых моделях и языках не поддерживается. Ниже предлагаются механизмы организации данных и методы управления вычислениями, расширяющие возможности асинхронного управления в СПД. Показывается, каким образом предлагаемые понятия расширяют функционально-потоковую модель параллельных вычислений и могут быть реализованы в языке программирования «Пифагор» [3−4].

Специфика асинхронного управления данными

Опираясь на работу Хоара [1], будем считать, что события, возникающие при выполнении параллельных процессов, порождаются последовательно. В СПД эти события связаны с моментами получения значений данных. Интервал времени между порождаемыми событиями в асинхронных моделях не считается существенным фактором. Более важную роль в данном случае играют соотношения между моментами порождения данных и длительностью вычислительных операций. Эти времена могут быть сравнимыми или соотноситься как очень большие и очень малые величины. Опираясь на представленные рассуждения, можно предложить следующую схему вычисления суммы элементов одномерного массива на основе готовности данных. Элементы массива, асинхронно формируемые в

ходе вычислений, образуют очередь, в которую заносятся в порядке появления. При наличии в этой очереди двух или более элементов проводится суммирование образуемых пар с последующей засылкой результата в эту же очередь. Единственный элемент, оставшийся в ходе суммирования, является окончательным результатом. Схема подобной организации вычислений представлена на рис. 4.

Для реализации такого управления в функциональную модель потоковых вычислений предлагается добавить асинхронный список. Его особенностью является упорядочение данных в соответствии с последовательностью их порождения. По сравнению с обычным списком данных, выдающим сигнал готовности после поступления всех его элементов, асинхронный список сигнализирует только о появлении первого элемента.

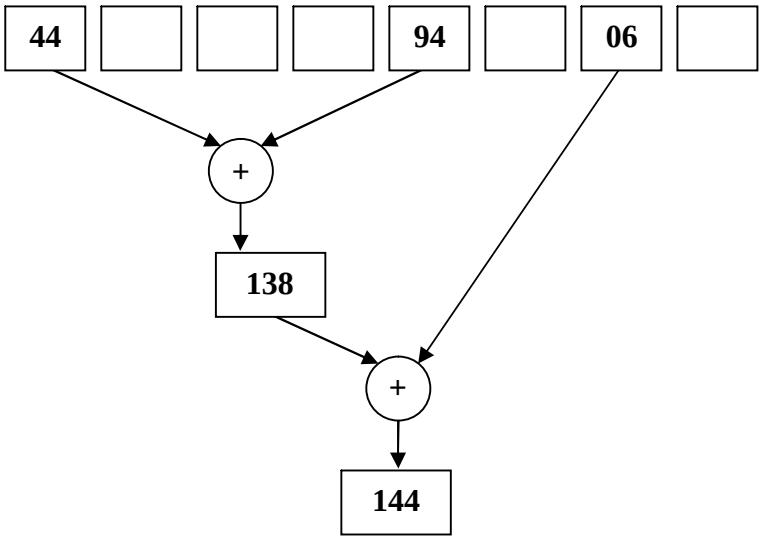


Рис. 3. Суммирование появившегося элемента с вычисленным промежуточным значением

Это накладывает определенную специфику на его интерпретацию. Можно считать, что об асинхронном списке достоверно известно следующее: или он не готов, или содержит как минимум один готовый элемент (а об остальных данных ничего не известно), или является пустым.

Неготовность асинхронного списка определяется отсутствием сигналов, информирующих о наличии сформированных внутри него данных. В этом случае, в соответствии с принципами потокового управления по готовности данных, такой список не может быть обработан. Выдача сигнала подтверждает готовность асинхронного списка к вычислениям, информируя при этом об одной из двух ситуаций: список содержит хотя бы один элемент или он пуст. Тогда определение внутреннего состояния возможно с использованием функции, определяющей длину асинхронного списка.

Длина пустого списка равна нулю. Длина непустого асинхронного списка всегда равна единице. Это объясняется спецификой его интерпре-

тации. Как только внутри списка появляется хотя бы один вычисленный элемент, происходит выдача сигнала, подтверждающего готовность списка к последующей обработке. Даже при одновременном порождении внутри асинхронного списка нескольких элементов будет выдаваться только один сигнал. В этой ситуации первый элемент списка определяется внутренним арбитром, стратегия работы которого не имеет существенного значения.

Наряду с определением длины, над асинхронным списком допустимы операции чтения первого элемента и формирования нового асинхронного списка, не содержащего первый элемент. Это, в соответствии с концепциями событийного управления, позволяет описывать асинхронный параллелизм как процесс обработки последовательно наступающих событий.

Чтобы отличить асинхронный список от обычного списка данных, ис-

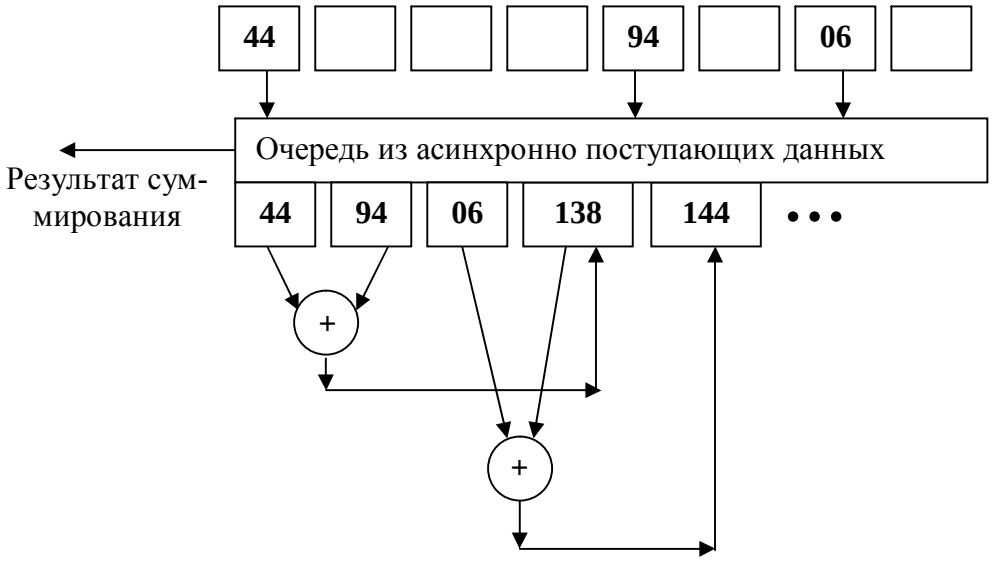


Рис. 4. Суммирование асинхронно поступающих данных

пользуемого в языке «Пифагор», введем для него следующее обозначение: asynch (d_1, d_2, … d_N), где d_1, d_2, … d_N − список элементов, порождаемых в ходе вычислений, проистекающих внутри него, упорядоченный по времени формирования значений. Для обработки этих значений допускается использовать операции выборки первого элемента и создания списка без первого элемента, эквивалентные по синтаксису и семантике, аналогичным операциям списка данных. Таким образом, выделения из списка первого элемента задается следующим образом:

asynch(d_1, d_2, … d_N):1 $\Rightarrow$ d_1 .

Для выделения прочих элементов в отдельный асинхронный список необходимо воспользоваться следующей операцией:

asynch(d_1, d_2, … d_N):-1 $\Rightarrow$ asynch(d_2, … d_N) .

Одновременное использование этих двух операций позволяет выделить и обработать все элементы асинхронного списка. Выполнение этих

операций может продолжиться до получения вместо асинхронного списка пустого списка данных:

asynch(d_1, d_2, d_3, … d_N):[1, -1] $\Rightarrow$ d_1, asynch(d_2, d_3, … d_N,) :[1, -1] $\Rightarrow$ $\Rightarrow$ d_1, d_2, asynch(d_3, … d_N,) :[1, -1] $\Rightarrow$ d_1, d_2, d_3, asynch(d_4, … d_N) :[1, -1] $\Rightarrow$ … … $\Rightarrow$ d_1, d_2, d_3, … d_{N-1}, asynch(d_N):[1, -1] $\Rightarrow$ d_1, d_2, d_3, … d_N, (.).

Прямой доступ к произвольному элементу асинхронного списка невозможен. Если же данные были сформированы заранее, то порядок их следования в асинхронном списке определяется размещением в исходном списке данных.

Дополнительные аксиомы и операции интерпретации функционально-потоковой модели вычислений

Для реализации асинхронного управления в функционально-потоковой модели вычислений [3, 4] введем ряд дополнительных понятий. Пусть имеется асинхронный список asynch (g_1, g_2, … g_n), где g_1, g_2, … g_n – генераторы данных, порождающие величины в произвольной последовательности. Как и список данных, асинхронный список фильтрует (отбрасывает, игнорирует) пустые элементы, не выставляя их очередь данных:

asynch(g_1, .) ≡ asynch(g_1) .

Асинхронные списки можно преобразовывать в другие списки. Преобразование в параллельный список осуществляется по следующему правилу:

asynch(d_1, d_2, … d_N):[] $\Rightarrow$ [d_1, d_2, … d_N] .

В результате на выходе формируется параллельный список, состоящий из множества независимых данных. Порядок следования данных в параллельном списке совпадает с последовательностью их порождения в асинхронном.

Операция преобразования асинхронного списка в список данных определяется аналогичным образом:

asynch(d_1, d_2, …, d_N):() $\Rightarrow$ (d_1, d_2, … d_N) .

Допустимы также обратные преобразования существующих списков в асинхронные, которые осуществляются с использованием существующих аксиом и правил преобразования модели вычислений. Непосредственным является только преобразование списка данных в асинхронный список:

(d_1, d_2, … d_N):asynch $\Rightarrow$ asynch(d_1, d_2, … d_N) .

При преобразовании параллельного и задержанного списков применяются аксиомы модели и языка, определяющие групповые операции [3, 4].

Использование асинхронных списков в языке программирования «Пифагор»

Приведенные выше расширения функционально потоковой модели вычислений можно реализовать в языке программирования «Пифагор». Это позволит писать параллельные программы в несколько ином стиле, при котором учитывается асинхронное поступление данных. В частности, сложение элементов одномерного массива, соответствующее схеме, представленной на рис. 4, может быть реализовано следующим образом:

```
// Функция, возвращающая сумму элементов асинхронного списка
A_VecSum<< funcdef A { // Формат A: asynch(x1, x2, … , xn)
  x1<< A:1;            // Поступивший элемент данных
  tail_1<< A:-1;       // Хвост асинхронного списка
  // Проверка на пустоту остатка списка
  [((tail_1:|, 0):[=, !=]):?]^
  (
     x1, // В списке, только один элемент,  определяющий сумму
     { // Выделение второго аргумента с последующим суммированием
        block {
           x2<< tail_1:1;       // второй аргумент
           s<< (x1,x2):+;       // сумма двух поступивших элементов
           tail_2<< tail_1:-1;  // "хвост" от "хвоста"
           // Рекурсивная обработка оставшихся элементов
           [((tail_2:|, 0):[=, !=]):?]^
           (
              s,
              { asynch( tail_2:[], s):A_VecSum }
           ):. >>break
        }
     }
  ):. >>return;
}
```

Следует отметить, что при поступлении на вход данной функции обычного списка данных вместо асинхронного списка он будет корректно обработан. Это обуславливается эквивалентной интерпретацией операций взятия первого элемента и выделения подсписка без первого элемента для обоих списков.

Интерпретация временных соотношений асинхронной потоковой программы

Использование асинхронных списков позволяет разрабатывать программу, которая, в зависимости от временных соотношений между ее операциями, может интерпретироваться как набор альтернативных алгоритмов, описывающих одну и ту же задачу. Например, если интервал времени Δt_{dat} между порождением данных внутри асинхронного списка больше времени выполнения операции сложения Δt_{add} (от момента чтения данных

из списка до засылки их во вновь формируемый асинхронный список), то суммирование будет выполняться последовательно. Данная ситуация проиллюстрирована на рис. 2 и 3.

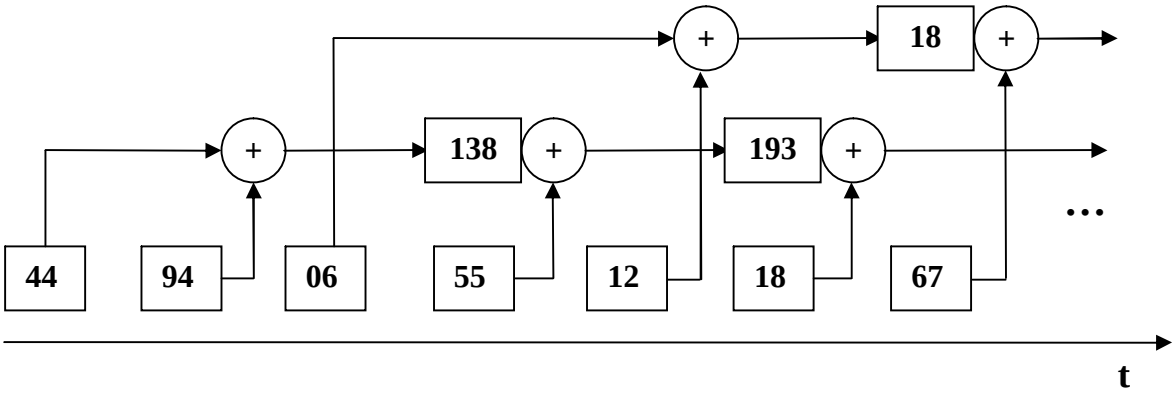


Рис. 5. Формирование двух параллельных потоков при $\Delta t_{add} = 2\Delta t_{dat}$

В том случае, если интервал времени поступления данных станет меньше времени их сложения, начнет появляться параллелизм в выполнении этих операций. Количество параллельных потоков будет кратно отношению времени Δt_{add} к Δt_{dat}. Например, при $\Delta t_{add}/\Delta t_{dat} = 2$ будет динамически сформировано два потока операций сложения (рис. 5).

Если темп поступления данных много выше времени выполнения операции сложения, то можно считать, что они успели поступить до начала суммирования, которое начнется практически одновременно для всех пар чисел. По завершении суммирования результаты также «мгновенно» поступают в асинхронный список, обеспечивая тем самым одновременное суммирование на втором слое. При этом общая схема вычислений будет соответствовать древовидной свертке, представленной на рис. 1.

Заключение

Использование асинхронных списков позволяет описать новый класс алгоритмов, параллелизм которых зависит от временных соотношений между выполняемыми основными и подготовительными операциями. Это ведет к специфическому подходу при построении параллельных алгоритмов и позволяет исследовать новые методы создания переносимых параллельных программ. В отличие от эквивалентных преобразований обобщенных функций, предлагаемых в [5] для адаптации параллельного алгоритма к конкретной вычислительной системе, использование асинхронных списков позволяет применять один и тот же алгоритм

Предложенный в работе механизм описания асинхронных вычислений предоставляет в распоряжение программиста дополнительные возможности. Его инструментальная поддержка на уровне языков обеспечивает использование новых методов разработки функционально-потоковых

параллельных программ, характеризующихся динамическим изменением параллелизма, в зависимости от временных соотношений между выполняемыми операциями. Это повышает гибкость разрабатываемых алгоритмов и позволяет использовать один и тот же алгоритм для динамического задания различных уровней параллелизма. Формируемые при этом программы позволяют описывать максимальный параллелизм решаемой задачи и не зависят от архитектур используемых вычислительных систем.

Литература

1. Хоар Ч. Взаимодействующие последовательные процессы / Пер. с англ. – М.: Мир, 1989. – 264 с.
2. Алгоритмы, математическое обеспечение и проектирование архитектур, многопроцессорных вычислительных систем / Под ред. А.П. Ершова. – М.: Наука, 1982. – 322 с.
3. Легалов А.И. Функциональный язык для создания архитектурно-независимых параллельных программ // Вычислительные технологии. − 2005. – Т.10, № 1. − С. 71–89.
4. Легалов А.И., Кузьмин Д.А., Казаков Ф.А., Привалихин Д.В. На пути к переносимым параллельным программам // Открытые системы. − 2003. – № 5. − С. 36–42.
5. Легалов А.И., Казаков Ф.А. Эквивалентные преобразования функционально-параллельных программ // Распределенные и кластерные вычисления: Избранные материалы Третьей школы-семинара / Институт вычислительного моделирования СО РАН. – Красноярск, 2004. – С. 134–141.

ПРОМЕЖУТОЧНОЕ ПРЕДСТАВЛЕНИЕ ФУНКЦИОНАЛЬНОГО ЯЗЫКА ПОТОКОВОГО ПРОГРАММИРОВАНИЯ

А.В. Редькин

КГТУ, Красноярск

Язык Пифагор является развивающимся языком функционального параллельного программирования. Он создан на основе функциональной потоковой модели [1], которая обеспечивает поддержку вычислений по готовности данных.

Модель вычислений, лежащая в основе языка Пифагор предполагает гибкое изменение стратегии управления для интерпретации программ в зависимости от условий, предоставляемых вычислительной и программной архитектурой. С другой стороны, информационный граф программы, его структуры и основные принципы выполнения по готовности данных не изменяются. Таким образом, для интерпретации программы необходимо выделить ее смысловую часть, описываемую моделью вычислений, а для вычислений использовать стратегию интерпретации, наиболее подходящую для решения поставленной задачи.

Промежуточное представление языка Пифагор разрабатывалось с целью четкого выделения информационного графа и всего необходимого для последующего наложения управляющего графа и интерпретации или компиляции. Каждая функция транслируется в промежуточное представление отдельно и для нее формируется массив внешних связей с типами и функциями. Промежуточное представление функции и ее связи помещаются в репозитарий, откуда среда интерпретации или компоновщик может запросить нужную функцию со всеми ее связанными функциями и типами (рис. 1).

На этапе трансляции функция проходит проверку на синтаксическую и семантическую правильность. На этапе помещения в репозитарий происходит проверка на наличие всех необходимых для выполнения функций. Таким образом, системе интерпретации не нужно делать лишних проверок, что экономит ценное время в ходе сборки и интерпретации программы.

Информационный граф для функции формируется в виде таблицы вершин, для каждой вершины в этой таблице существует таблица связанных с ней вершин, то есть вершин, результаты вычисления которых используются данной вершиной. Соответственно, константы не имеют свя-

занных вершин, узлы интерпретации имеют две связанные вершины, списки в качестве связанных вершин имеют составляющие их элементы.

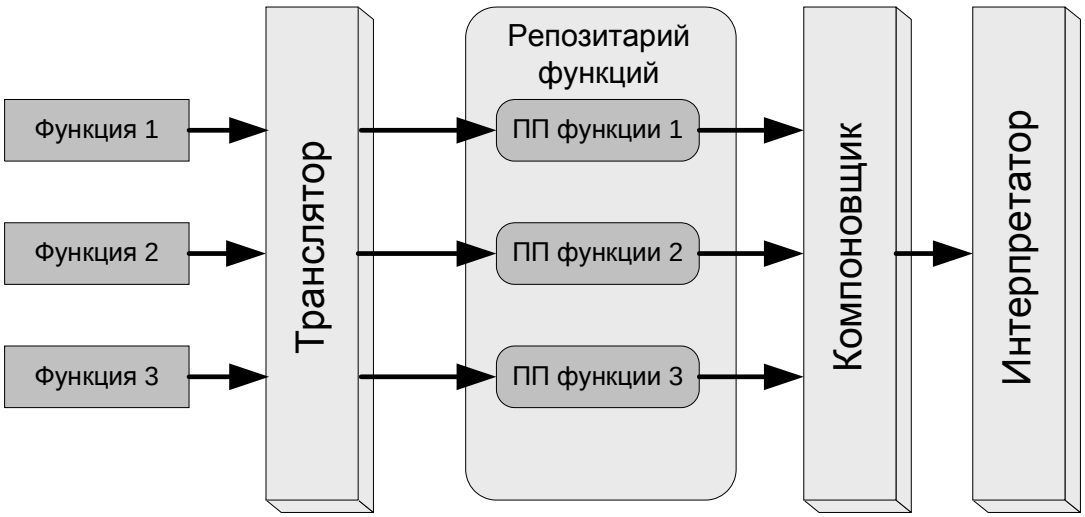


Рис.1. Схема использования промежуточного представления

Связывание вершин производится на основе уникальных идентификаторов. На рис. 2 представлен информационный граф выражения [(x,1),(x,2)]:+, а на рис. 3 схематично представлено, как будет выглядеть промежуточное представление такого графа.

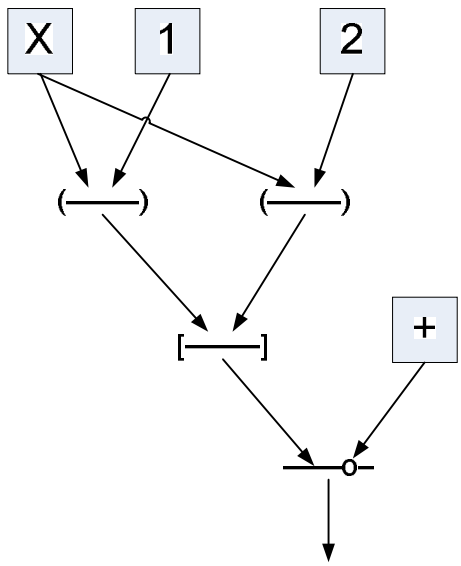


Рис. 2. Информационный граф
выражения [(x,1),(x,2)]:+

Создание промежуточного представления языка Пифагор является первым шагом к поддержке модульных программ. Также данная разработ-

ка предполагает создание трансляторов и систем исполнения программ, совместимых на уровне промежуточного представления.

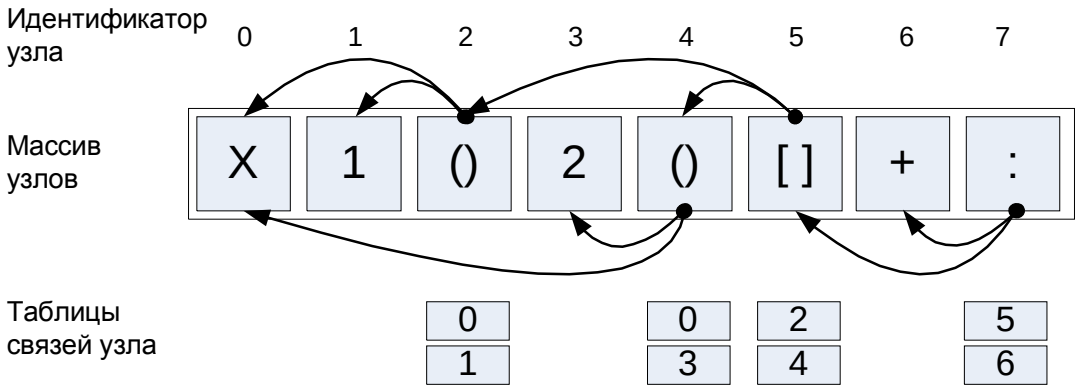


Рис. 3. Представление информационного графа в виде массива узлов

Современная поддержка модульного программирования, как правило, включает возможности использования глобальной коллекции функций или классов, доступных компилятору и системе интерпретации. Популярные системы переносимого программирования, такие как Java и Microsoft.NET, предоставляют программистам такие возможности. Использование глобального хранилища функций на компьютере или в сети позволяет упростить разработку, компиляцию и исполнение модульных программ.

Репозитарий функций в общем случае позволяет организовать модульность программ независимо от используемых систем трансляции и интерпретации и исполнения программ, что схематично изображено на рис. 4.

В качестве эксперимента разработаны структуры промежуточного представления и библиотека для работы с ним, разработана структура репозитария функций, реализован транслятор из языка Пифагор. Во всех программных разработках использован стандартный язык C++, что позволяет говорить о хорошей переносимости программ на многие архитектуры.

Промежуточное представление позволяет упростить создание систем интерпретации, а также является основой для поддержки модульного программирования. Работа в дальнейшем позволит более детально описать модель вычислений, ее динамику функционирования и стратегии управления.

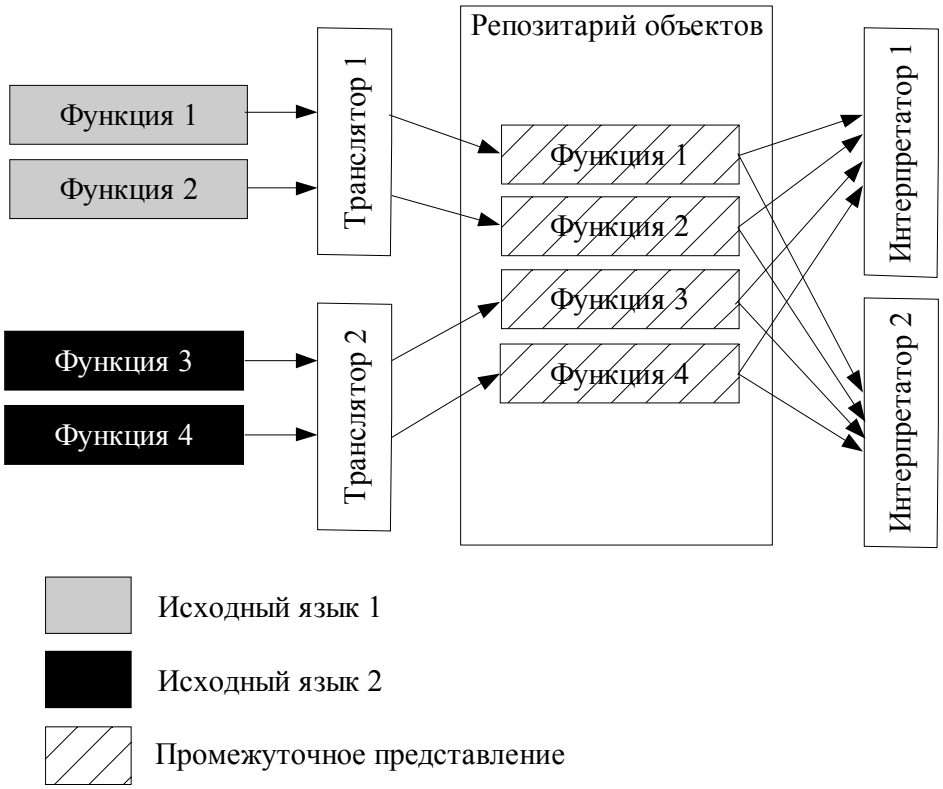


Рис. 4. Использование репозитария

В настоящее время работа над промежуточным представлением потокового языка параллельного программирования продолжается, разрабатывается представление таких элементов, как асинхронный список, которые позволят более точно описать информационный граф.

Литература

1. Легалов А.И. Функциональный язык для создания архитектурно-независимых параллельных программ // Вычислительные технологии. − 2005. − Т.10, № 1. − С. 71−89.
2. Редькин А.В. Промежуточное представление информационного графа для языка Пифагор // Информатика и информационные технологии: Материалы межвузовской научной конференции. − Красноярск: ИПЦ КГТУ, 2004. − С. 52.

МОДЕЛЬ ВЫЧИСЛИТЕЛЬНЫХ РЕСУРСОВ С НЕОДНОРОДНОЙ КОММУТАЦИЕЙ

Е.А. Гнедов

КГТУ, Красноярск

Введение

Использование параллельных вычислительных систем дает неоценимые преимущества при решении больших вычислительных задач, но при этом возникает целый ряд трудностей, связанных, с одной стороны, с разработкой параллельных алгоритмов, а с другой – с управлением ресурсами параллельной вычислительной системы (ПВС).

Следует заметить, что общий характер проблем, сопровождающих развитие параллельных вычислений, в целом выглядит таким же, каким он был и во времена последовательных. Только для параллельных вычислений они проявляются в более острой форме во многом из-за большей сложности самой предметной области.

Ситуация обостряется наличием многообразия в различных параллельных архитектурах, что в свою очередь породило множество методов написания параллельных программ. Во многом это было обусловлено тем, что языки программирования непосредственно разрабатывались под особенности ПВС. Такое программирование, несмотря на эффективное решение задач в рамках конкретной архитектуры, привело к проблемам при переносе программного обеспечения. Однако появление универсальных языков параллельного программирования тоже мало способствовало улучшению ситуации в сфере как разработки алгоритмов, так и управления ресурсами ПВС, так как универсальность в данном случае оказалась весьма многогранной.

Исходя из того, что в настоящее время наблюдается постоянное повышение мощности параллельных вычислительных систем, существуют ПВС, состоящие из десятков тысяч процессоров, можно сделать выводы о том, что актуальной задачей является разработка новых методов управления вычислительными ресурсами и данными этих ПВС.

Используемые подходы, в основном, ориентированы на статическое распределение ресурсов ПВС, что при повышении размерности решаемой задачи ведет к несбалансированности вычислений и повышению нагрузки на отдельные элементы ПВС. Положение усугубляется тем, что разрабатываемые вычислительные системы используют различные внутренние методы обмена данными между параллельными процессами [1].

На данный момент можно выделить два основных направления в управлении ресурсами параллельных вычислительных систем: статическое

управление и динамическое управление ресурсами. В основе статического распределения ресурсов лежит модель, реализующая выполнение параллельных процессов на заранее выделенных ресурсах ПВС, набор которых в течение времени выполнения программы остается постоянным. При этом наблюдаются следующие основные негативные факторы: во-первых, ограниченная масштабируемость программы, а именно, при изменении характеристик ПВС в сторону увеличения или уменьшения доступных ресурсов приходится с большой долей вероятности изменять программу. Во-вторых, управление ресурсами ПВС находится в руках разработчика и именно от него в большей степени зависит, будет ли эффективно выполняться программа на доступных ресурсах ПВС или большую часть времени процессы будут простаивать в ожидании данных.

Более гибкое управление ресурсами параллельных вычислительных систем было бы возможно при использовании динамического управления ими. Однако в настоящее время эти методы не обеспечивают эффективного управления (загрузки и балансировки нагрузки). Следует также отметить отсутствие эффективных и обобщенных моделей динамического распределения ресурсов, что во многом объясняется разнообразием механизмов порождения вычислительных процессов и методов обмена данными между ними.

Методы реализации параллелизма

Современные параллельные вычислительные системы являются симбиозом различных комбинаций параллельных архитектур, используемых при взаимодействии процессов. Множество таких архитектур можно разделить на два класса: параллельные архитектуры и псевдопараллельные архитектуры. Последний вид на уровне процессора использует последовательное выполнение, а пользователю предоставляется параллельная виртуальная архитектура.

Рассмотрим методы управления параллельными процессами и технологию обмена данными в этих классах архитектур:

1. Параллельные вычислительные архитектуры:

Конвейерные процессоры

Специализированные процессоры, выполняющие последовательность определенных действий над множеством данных. Со стороны процессора данные исполняются параллельно, но со стороны пользователя происходит последовательное выполнение программы.

Многоядерность (CoreMultiProcessor – CMP-процессор)

Принцип многоядерной архитектуры – размещение двух или более «исполняющих» ядер в одном процессоре. CMP-процессор вставляется в один процессорный разъем, но операционная система компьютера воспринимает каждое из его исполняющих ядер как отдельный логический про-

цессор, обладающий всеми соответствующими исполняющими ресурсами. Разделяя вычислительную работу, выполняемую в традиционных микропроцессорах одним ядром, между несколькими исполнительными ядрами, многоядерный процессор может выполнять больше работы в единицу времени.

Кластеры

Независимые вычислительные процессоры, имеющие свою память, объединенные с помощью достаточно медленных, по сравнению с обменом через общую память, коммутационных каналов (обычно – Ethernet).

2. Псевдопараллельные вычислительные архитектуры:

Мультипрограммный режим

Поддерживаемый операционной системой механизм управления разделением доступа выполняющихся процессов к процессорному времени путем переключения их контекста.

Многопоточность (Multithreads)

Более легкий вариант мультипрограммного режима. Части кода процесса могут выполняться как отдельные процессы в рамках контекста основного процесса. Доступ к процессорному времени так же, как и в мультипрограммном режиме, квантуется.

Взаимодействие процессов в различных архитектурах может происходить с помощью различных методов: последовательная передача данных (конвейерная обработка данных), взаимодействие через общую память (мультипрограммный или многоядерный режимы) или обмен сообщениями (кластеры) и т.д.

Различные методы управления процессами, а также взаимодействия между процессами в различных параллельных архитектурах и тем более совместное их использование в вычислительных системах приводят либо к созданию управляющих механизмов для какого-то одного вида архитектуры, либо к разработке и реализации таких методов управления, которые бы выбирали наилучший вариант управления ресурсами из возможно допустимых, и решали такие актуальные задачи, как платформонезависимость, масштабируемость, балансировка нагрузки, анализ процесса выполнения и т.д., зачастую возникающие и являющиеся препятствиями, усложняющими разработку параллельных программ при использовании методов статического управления ресурсами параллельных вычислительных систем.

В данной работе делается попытка унифицировать и представить механизмы обмена данными между процессами с единой точки зрения, а именно предложить модель управления ресурсами ПВС, в основании которой лежат два предположения: первое – все узлы однородны, второе – узлы, связанные коммутационными каналами различной пропускной способности.

Описание модели

Для того чтобы сделать структуру параллельной вычислительной системы более прозрачной для разработчика, т.е. предложить взаимодействие не с физической ПВС, которая может состоять из различных параллельных архитектур и иметь множество методов для обмена данными между процессами, тем самым, сводя разработку к решению сразу двух задач, а именно: к разработке алгоритма и к разработке стратегии выделения необходимых ресурсов ПВС, а предложить взаимодействие с обобщенной или унифицированной моделью параллельной вычислительной системы, которая, с одной стороны, позволяла бы разрабатывать платформонезависимые программы и, с другой стороны, являлась бы эффективным механизмом по управлению и распределению ресурсов ПВС и поддержанию различных параллельных архитектур и методов обмена данными.

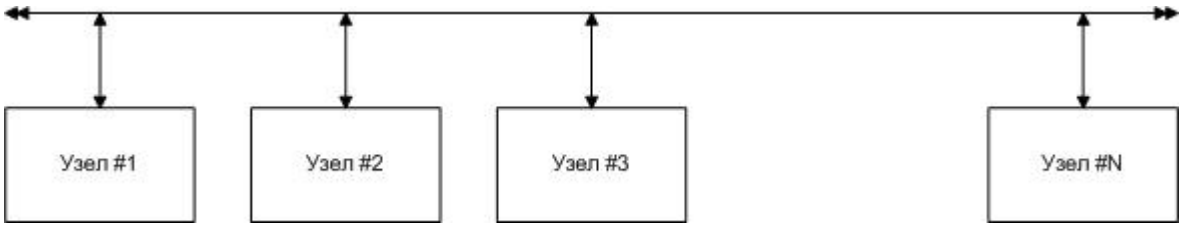


Рис. 1. Обобщенная модель ПВС с однородными узлами

Обобщенная модель параллельной вычислительной системы (рис. 1), которая в нашем случае базируется на однородной ПВС (например, кластер, который с большой долей вероятности будет однородным), основывается на предположении о том, что процессорные узлы являются однородными и взаимодействуют между собой по неоднородным коммутационным каналам.

Каждый вид каналов ассоциируется с тем или иным методом физического обмена данными. К различным видам неоднородных каналов относятся следующие методы взаимодействия процессов: последовательное взаимодействие между вызывающей и вызываемой процедурой, взаимодействие между независимыми потоками в рамках мультипрограммной среды, взаимодействие между параллельно выполняемыми процессами через общую память и т.д.

В конечном итоге обобщенная параллельная вычислительная система с обозначенными коммутационными каналами представлена на рис. 2, где физический узел №1 может, например, поддерживать параллелизм на уровне ядра процессора, следовательно, его можно представить как два ресурса, которые имеют более производительный коммутационный канал

для обмена данными, т.е. для программы это будут разнозначные узлы, но доступ к ним осуществляется по более быстрому каналу передачи данных.

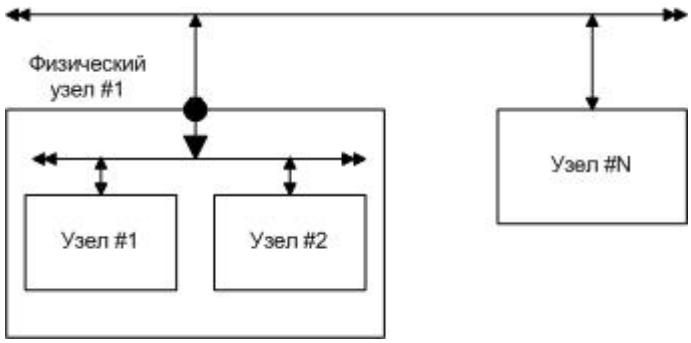


Рис. 2. Модель ПВС с неоднородными
коммутационными каналами

Выбор той или иной стратегии управления ресурсами ПВС зависит от нескольких факторов:

1) пропускной способности коммутационных каналов;
2) доступных ресурсов ПВС;
3) характеристики параллельного алгоритма.

Каждый вид коммутационных каналов вносит свои задержки в процесс обмена данными между процессами, эти задержки можно классифицировать следующим образом:

1. Задержки на узле параллельной вычислительной системы.
2. Задержки при взаимодействии между узлами (в контексте обобщенной модели, т.е. взаимодействие не только между процессорными узлами, а обмен данными по любым коммутационным каналам) ПВС.

К задержкам первого вида относятся следующие: выполнение кода функции, порождение контекста процесса, загрузка данных для обработки, ожидание момента запуска.

Задержки взаимодействия между узлами в основном связаны с физической пропускной способностью каналов, если речь идет об обмене с помощью Ethernet или внутренними задержками на процессорном узле, которые связаны, например, с синхронизацией доступа к общей памяти или ожидания переключения контекста процесса в мультипрограммной среде.

В дополнение к этому следует также отметить наличие задержки на принятие решения управляющим процессом о запуске того или иного процесса и миграции данных с одного узла на другой.

Процесс выполнения параллельной программы разбивается на несколько этапов, а именно: первый этап – оценка ресурсов ПВС и оценка

информационного графа на предмет определения степени используемого параллелизма, второй – принятие решения об использовании той или иной стратегии управления ресурсами ПВС и третий – непосредственный запуск программы, в ходе которого происходит выполнение функций, а также принятие решения о следовании выбранной стратегии, либо принятия иной, в случае, если состояние системы изменилось настолько, что далее невозможно эффективное исполнение программы.

В момент запуска программы каждый узел параллельной вычислительной системы содержит в себе каждую функцию и все начальные данные, и становится понятно, что стратегия управления – это есть не что иное, как метод обхода информационного графа и принятие решения о том, на каком узле будут выполняться те или иные функции и необходимо ли для этого передавать какие-нибудь данные.

Из всего вышесказанного мы приходим к выводу, что использование обобщенного представления различных физических методов обмена данными посредством одинаково устроенных виртуальных каналов с различными временными параметрами, вносящими задержки при обмене данными, позволяет обобщить процесс анализа методов динамического выделения вычислительных ресурсов для программ, не учитывающих на этапе разработки методов выделения вычислительных ресурсов.

Заключение

Предложенная модель в полной мере описывает взаимодействие виртуальной параллельной вычислительной системы, реализующей модель максимального параллелизма и реальной ПВС с ограниченными ресурсами.

В результате мы получаем новый подход в разработке параллельных программ – написание алгоритмов в машинно-независимой форме с последующей их адаптацией к конкретной вычислительной среде путем сжатия максимального параллелизма с учетом налагаемых ресурсных и функциональных ограничений.

Учитывая все вышесказанное, мы можем проводить анализ алгоритмов с точки зрения эффективности их выполнения на параллельных вычислительных системах, а также проводить исследования в сфере управления ресурсами ПВС в различных ситуациях, вызываемых постоянными изменениями в состоянии ресурсов вычислительной системы.

Данный подход предполагается использовать для динамического распределения ресурсов в программах, написанных на функционально-потоковом языке параллельного программирования Пифагор [2], оценки эффективности моделей управления распределением вычислительных ресурсов параллельных вычислительных систем, а также создания имитационного комплекса для проведения исследований взаимодействия парал-

лельных процессов, исполняемых в условиях ограниченных ресурсов различных по архитектуре параллельных вычислительных систем.

Литература

1. Воеводин В. В. Математические проблемы параллельных вычислений // Сб. тр. Всероссийской конф. «Научный сервис в сети Интернет: технологии распределенных вычислений». Новороссийск, 19-24 сентября 2005 г. – М.: Изд-во МГУ, 2005. – С. 3−8.
2. Легалов А. И., Казаков Ф. А., Кузьмин Д. А., Привалихин Д. В. Функциональная модель параллельных вычислений и язык программирования «Пифагор». http://softcraft.ru/parallel/fpp/fppcontent.shtml

МАТЕМАТИЧЕСКИЕ ПРОБЛЕМЫ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ*

В.В. Воеводин

ИВМ РАН, Москва

Возможность быстрого решения задач на вычислительной технике параллельной архитектуры вынуждает пользователей изменять весь привычный стиль взаимодействия с компьютерами. По сравнению, например, с персональными компьютерами и рабочими станциями меняется практически всё: применяются другие языки программирования, видоизменяется большинство алгоритмов, от пользователей требуется предоставление многочисленных нестандартных и трудно добываемых характеристик решаемых задач, интерфейс перестает быть дружественным и т.п. Важным является то обстоятельство, что неполнота учета новых условий работы может в значительной мере снизить эффективность использования новой и к тому же достаточно дорогой техники.

Надо заметить, что общий характер трудностей, сопровождающих развитие параллельных вычислений, в целом выглядит таким же, каким он был и во времена последовательных. Только для параллельных вычислений все трудности проявляются в более острой форме. Во многом из-за большей сложности самой предметной области. Но, возможно, главным образом вследствие того, что к началу активного внедрения вычислительных систем параллельной архитектуры в практику решения больших прикладных задач не был построен нужный теоретический фундамент и не был развит математический аппарат исследований. В конце концов, из-за этого оказался своевременно не подготовленным весь образовательный цикл в области параллельных вычислений, отголоски чего проявляются до сих пор. Отсюда непонимание многочисленных трудностей освоения современной вычислительной техники, пробелы в подготовке нужных специалистов и многое другое.

Теоретический фундамент. Последовательные вычисления развивались не одну сотню лет. За это время пришло довольно чёткое понимание, что такое последовательный алгоритм. Вокруг данного понятия сформировался большой раздел математики, называемый теорией алгоритмов, изучающий общие свойства последовательных вычислений. Уточнённое понятие алгоритма в терминах идеализированных вычислительных машин

* Сстатья перепечатана из сборника трудов Всероссийской конференции «Научный сервис в сети Интернет: технологии распределенных вычислений». Новороссийск, 19–24 сентября 2005 г. – М.:Изд-во Моск. ун-та, 2005. – С. 3–8.

привело к очень важному понятию машины Тьюринга. По существу этот автомат стал теоретическим прообразом первых ЭВМ. Присоединение к машине Тьюринга памяти сделало её весьма полезным и даже приближенным к реальности инструментом исследований. Но в основе всего лежали последовательные действия. Не удивительно поэтому, что и ЭВМ в течение длительного периода также развивались по пути реализации именно последовательных действий.

Хотя общая направленность последовательного выполнения операций сохранялась довольно долго, в разработке вычислительной техники незаметно назревали *революционные изменения*, приведшие, в конце концов, к радикальному пересмотру всех представлений о вычислениях. Причиной возникновения этих изменений стал параллелизм, внедряемый в вычислительную технику во имя повышения производительности. Пока параллельное выполнение в компьютере любых операций, передач информации и обращений к памяти не приводило к принципиальным изменениям в языках программирования, у пользователей не было особых причин думать о параллелизме. Но в определённый момент параллелизма в компьютере стало столь много, что его присутствие уже нельзя было прикрывать техническими решениями. И тогда от пользователя стали требовать предоставления дополнительной информации о структуре используемых им алгоритмов, требовать как раз для того, чтобы эффективно использовать заложенный в компьютер параллелизм. Однако сам пользователь оказался к выполнению этих требований не готов.

Очень скоро выяснилось, что *пользователь знает эту информацию далеко не всегда.* Более того, чаще всего он даже не понимает, откуда и как ее получать. И оказалось, что добывается она, как правило, с большим трудом.

К началу массового внедрения вычислительных систем параллельной архитектуры многие математические вопросы параллельных процессов оказались в зачаточном состоянии. Не было никакой целостной теории параллельных алгоритмов, аналогичной теории алгоритмов для последовательных вычислений. Существовали лишь отдельные разрозненные результаты. Не было даже сколько-нибудь ясного представления, что же нужно понимать под параллельным алгоритмом. И, конечно, отсутствовал какой-либо формальный математический аппарат, который можно было бы назвать параллельным аналогом машины Тьюринга.

Скорее всего, именно эти причины привели к тому, что в течение долгого времени параллельные вычисления не удавалось сформировать как самостоятельную математическую науку, и рассматривались они как совокупность каких-то полуэвристических, граничащих с искусством приемов приспособления алгоритмов к требованиям новой техники.

Новые параллельные алгоритмы. Несмотря на все это, параллельные алгоритмы начали создаваться уже давно. Значительный интерес к их

построению на основе математически эквивалентных преобразований возник в 60 – 70-х годах прошлого столетия в связи с появлением первых вычислительных систем параллельной архитектуры.

Чтобы оценить время реализации алгоритма на параллельной системе, алгоритм представляют в виде последовательно выполняемых ансамблей операций, причём в каждом ансамбле все операции не должны быть связаны друг с другом. Если архитектура параллельной системы позволяет реализовывать одновременно все операции каждого ансамбля, то без учёта времени на передачи данных время выполнения алгоритма будет пропорционально числу ансамблей. Число ансамблей стали называть *высотой* алгоритма. Алгоритмы, в которых высота меньше общего числа операций, стали называть *параллельными,* а их представление через последовательность ансамблей из независимых операций – *параллельной формой.*

Очевидно, что в зависимости от структуры связей между операциями один и тот же алгоритм может быть представлен различными способами в виде совокупности ансамблей. В частности, обычная последовательная реализация означает, что в каждом ансамбле содержится только одна операция. Для большинства алгоритмов даже таких представлений может существовать очень много. Ясно, что для каждой задачи особый интерес представляет нахождение алгоритмов *минимальной* высоты. Согласно теории последовательных алгоритмов представления *одного и того же алгоритма* различными ансамблями необходимо рассматривать как *разные алгоритмы*, так как изменяется, как минимум, порядок выполнения операций. Следовательно, некоторые характеристики этих разных алгоритмов окажутся заведомо различными, но какие-то наверняка сохранятся.

Так что же меняется и что сохраняется в алгоритмах при тех или иных преобразованиях? На все подобные вопросы требуется дать четкие математические ответы. Это необходимо сделать еще и потому, что различные программы, для создания которых авторами часто используется *один и тот же* алгоритм, на самом деле почти всегда описывают *разные* алгоритмы, хотя и математически эквивалентные. А это приводит на практике к разным результатам.

Чтобы задача построения быстрых параллельных алгоритмов стала математически корректной, необходимо сделать какие-то предположения относительно свойств параллельной вычислительной системы. Они очень просты: система имеет бесконечно много параллельно работающих процессоров; все они работают синхронно под общим управлением и выполняют любую операцию точно и за одно и то же время; система имеет бесконечно большую память; все обмены информацией между процессорами и памятью, а также между самими процессорами осуществляются мгновенно и без конфликтов. Концепция построения алгоритмов для подобных параллельных систем получила название *концепции неограниченного па-*

раллелизма. Конечно, она идеализирована. Тем не менее полученные в её рамках результаты интересны и поучительны.

Рассмотрим обычный процесс суммирования n чисел, когда на каждом шаге к частичной сумме прибавляется очередное слагаемое. Этот алгоритм имеет только одну параллельную форму, в каждом ансамбле которой имеется лишь одна операция. Следовательно, никакой возможности использовать параллелизм *в этом алгоритме* нет. Поскольку операция суммирования большого числа слагаемых является очень распространённой, был придуман другой способ суммирования, обладающий лучшим параллелизмом. Разобьём все слагаемые на пары и осуществим суммирование двух чисел внутри каждой пары. Все эти операции независимы. Полученные частные суммы также разобьём на пары и снова осуществим суммирование двух чисел внутри каждой пары. Снова все операции независимы. Вся сумма будет получена через $\log_2 n$ шагов. Это и будет высота *нового* алгоритма. В нём уже имеется значительный ресурс параллелизма, хотя он не равномерен. На первом временном шаге может быть использовано $n/2$ процессоров, на втором $n/4$ и т.д. Назван новый алгоритм процессом *сдваивания.*

Заметим, что оба алгоритма основаны на реализации математически эквивалентных выражений суммирования чисел, но они имеют разные свойства, по крайней мере, с точки зрения параллельных вычислений. На самом деле у них много и других различий: они по-разному реагируют на ошибки округления, по-разному используют память и т.п. Поэтому эти алгоритмы следует считать *принципиально различными*, несмотря на то, что они *математически эквивалентны*!

Пусть какой-то алгоритм существенно зависит от n входных данных и реализуется через некоторую совокупность операций, имеющих не более p аргументов. Легко показать, что такой алгоритм не может иметь высоту меньше, чем $\log_p n$. Очевидно также, что высота любого алгоритма ограничена сверху общим числом выполняемых операций. Эти две границы являются ориентирами для построения алгоритмов минимальной высоты. Например, сразу становится ясно, что суммирование чисел по принципу сдваивания относится к оптимальным алгоритмам.

Легко построить алгоритм наименьшей высоты для задачи умножения матрицы размера $n?m$ на вектор размера m. Компоненты вектор-результата могут быть вычислены независимо, и каждая из них определяется только $2m$ входными данными. Поэтому оценка высоты снизу должна иметь порядок $\log_2 m$. Соответствующий алгоритм получается очевидным образом на основе суммирования по принципу сдваивания. Задачу вычисления произведения двух матриц порядка n можно рассматривать как задачу вычисления n произведений одной и той же матрицы и n независимых векторов порядка n. Если все эти произведения вычислять независимо по опи-

санному правилу, то полученный алгоритм будет иметь высоту порядка $\log_2 n$.

Рассмотренные задачи исключительно просты, и построение для них алгоритмов наименьшей высоты не вызывает никаких трудностей. Но это скорее исключения, чем правило. Другие задачи оказываются значительно сложнее. Например, задача обращения плотной квадратной матрицы порядка n. Всего имеется n^2 входных данных, и каждый элемент обратной матрицы в общем случае существенно зависит от всех элементов исходной матрицы. Согласно сказанному ранее, оценка снизу минимальной высоты алгоритма обращения матрицы имеет порядок $\log_2 n$. В настоящее время построены алгоритмы с высотой порядка $\log_2^2 n$, и не известно, существуют ли алгоритмы существенно меньшей высоты. Аналогичное положение имеет место по отношению ко многим другим задачам: построены алгоритмы высоты, существенно меньшей, чем общее число операций, но *не известно*, можно ли эти алгоритмы улучшить по высоте.

Высота алгоритма является очень важной характеристикой, так как показывает потенциальную возможность быстрого решения задачи на вычислительной системе параллельной архитектуры. Однако пока параллельные алгоритмы малой высоты не вошли в практику использования сколько-нибудь широко. Причина очень проста: подавляющее большинство из них требует огромного числа процессоров, имеет сложные коммуникационные связи и катастрофически неустойчиво. Например, некоторые быстрые алгоритмы обращения матрицы размера $n?n$ требуют порядка n^4 процессоров. На самых современных системах этими методами можно обращать матрицы не более 10-го порядка, да и то лишь теоретически. На практике даже такие задачи будут решаться очень долго из-за исключительно сложных передач данных. Среди всех быстрых параллельных алгоритмов заметным исключением являются только суммирование чисел по принципу сдваивания и некоторые его аналоги. Подобные алгоритмы используются на практике достаточно широко.

Несмотря на отмеченные недостатки, концепция неограниченного параллелизма оказалась исключительно живучей. Предельная абстрагированность от реалий вычислительной техники сделала её привлекательной для математиков. Тем не менее на сегодняшний день все достижения в рамках этой концепции скорее представляют *набор отдельных изобретений* в области численных методов, чем систематически развивающийся раздел математики. Вполне возможно, что здесь ещё *не сказано последнее слово,* и к построению быстрых параллельных алгоритмов всё же будет разработан систематизированный подход, приводящий к более эффективным решениям.

Заметим, что практически все быстрые параллельные алгоритмы на самом деле могут рассматриваться как результат математически эквивалентных преобразований формульных выражений, описывающих хорошо

известные последовательные алгоритмы. При этом набор допустимых преобразований очень прост: ассоциативность, коммутативность, дистрибутивность, приведение подобных членов, а также замена нулевого слагаемого разностью, а единичного множителя отношением любых одинаковых выражений. А какой разброс в алгоритмических свойствах!

Ошибки округления. До сих пор мы рассматривали различные изменения свойств алгоритмов при математически эквивалентных преобразованиях. Основой таких преобразований было предположение о *точном* выполнении операций. Однако на всех без исключения компьютерах на представление любого числа отводится только *конечное,* строго фиксированное число разрядов. Поэтому после выполнения каждой операции результат «обрезается» до нужной длины. Эта процедура вносит в результат ошибку, которая называется *ошибкой округления*.

Сами по себе ошибки округления отдельных операций *очень малы*. Малы настолько, что часто возникает соблазн не учитывать их влияние на общий результат. К этому подталкивает и то обстоятельство, что во всех языках программирования любые формульные выражения записываются как математические без какого-либо указания на наличие ошибок округления. Более того, отметим как факт, что ни одна используемая на практике стандартная программная среда не имеет *инструментальных средств* для контроля за распространением этих ошибок. А это распространение происходит.

Важнейшим фактором, объясняющим влияние ошибок округления компьютерных операций на окончательный результат, является радикальное изменение свойств математических операций. Именно на множестве чисел, представленных в компьютере в форме с плавающей запятой, все операции *перестают обладать свойствами* коммутативности, ассоциативности и дистрибутивности. Аналогичная потеря свойств происходит и для чисел с фиксированной запятой, но в несколько меньшей мере. Из сказанного следует исключительно важный вывод: записывая в программах математически эквивалентные выражения, *мы не должны поддаваться иллюзии*, что эти программы будут давать на компьютере хотя бы похожие результаты. Известно, что даже такая простая операция, как перестановка слагаемых в суммах чисел, может привести из-за ошибок округления к катастрофически большим различиям.

Имеется много других задач, аспектов и вопросов, связанных с заменой одних формульных выражений другими, математически эквивалентными. Обратим внимание на следующие моменты. Во-первых, сфера замен формульных выражений исключительно обширна. Во-вторых, в теории и практике осуществления замен остаётся очень много белых пятен. И, наконец, даже если замены делаются математически эквивалентными, это ещё не гарантирует, что на практике мы не встретимся с большими неожиданностями.

Информационная структура алгоритмов. Вычислительные эксперименты показывают, что практически все *новые* параллельные алгоритмы, даже те из них, которые очень эффективны в теоретическом отношении, на практике не конкурентоспособны. Поэтому на текущий момент единственно надёжным источником создания параллельных программ является подходящая *реструктуризация* проверенных временем последовательных программ и математических описаний. Выбор этих форм записей объясняется тем, что только они позволяют описать алгоритмы более или менее точно.

Формально реструктуризация сводится к *математически эквивалентным* заменам в записях всех или части формульных выражений с целью явно указать обнаруженные в алгоритмах скрытый параллелизм, возможность использования распределенной памяти и т.п. Уже отмечалось, насколько внимательно нужно относиться к таким заменам. Снова ключевым моментом становится контроль над влиянием ошибок округления на результат. Ясно, что нужно иметь эффективные технологии как для выявления требуемых свойств алгоритмов, так и для выполнения преобразований самих записей к виду, в котором все эти свойства можно описать с помощью специальных комментариев. Очень важно, чтобы все такие технологии были максимально независимы от пользовательских знаний, касающихся решаемых задач и используемых алгоритмов.

Рассмотрим все *математически эквивалентные* записи какого-либо алгоритма. Пусть каждая из них сделана на своем языке и реализуется на своем компьютере. Будем лишь считать едиными правила приближенного выполнения операций над числами. Среди указанных записей заведомо существует какое-то множество, которое для одних и тех же входных данных будет давать при реализации один и тот же результат с учетом влияния всех ошибок округления. Естественно предположить, что у всего этого множества должно быть какое-то общее ядро. И тогда возникают вопросы, как оно выглядит, как его находить, как использовать и т.п.

Чтобы найти общее ядро, необходимо, прежде всего, очистить записи от всех языковых наслоений. После такой очистки остается лишь некоторая совокупность выполняемых операций, связанных между собой отношениями «результат–аргумент». Это задает граф, получивший название *граф алгоритма*. Можно показать, что для того чтобы в одинаковых условиях разные записи алгоритмов приводили к одним и тем же результатам, необходимо и достаточно, чтобы были *изоморфны* их графы. Построенные графы описывают информационные сущности алгоритмов. Они не зависят ни от используемых языков описания, ни от применяемых вычислительных средств. Поэтому вполне естественно их считать *информационными ядрами* самих алгоритмов.

Граф алгоритма имеет очень прозрачный смысл. Поэтому его легко применять в теоретических исследованиях. Однако чтобы этот граф ис-

пользовать в реальных приложениях, он должен быть явно задан в какой-либо форме, приемлемой для таких целей. На практике он никогда не бывает известен в нужном виде, и граф алгоритма приходится находить с помощью специальных методик из описывающих сам алгоритм программ или математических соотношений. Эти методики чрезвычайно сложны, и на их разработку ушло много лет. При этом пришлось поставить и решить довольно много новых и нетрадиционных математических проблем.

Отметим одну проблему, не очень заметную на первый взгляд, которая, тем не менее, на всех этапах исследований вызывала огромнейшие трудности. В графе алгоритма столько вершин, а по порядку и столько же дуг, сколько выполняется элементарных машинных операций за время реализации алгоритма. Для задач, решаемых на компьютерах часами и днями, их число настолько велико, что для прямого описания всех вершин и дуг соответствующего графа не хватит памяти самого большого компьютера в мире. Но ведь граф алгоритма нужно не только описать, его нужно и анализировать. В частности, необходимо находить различные ориентированные разрезы, критический путь, какие-то подграфы и т.п. Все такие задачи имеют полиномиальную или даже экспоненциальную сложность. Следовательно, при прямом задании графа любой его анализ, скорее всего, займёт гораздо больше времени, чем решение исходной задачи. Такой анализ практически бесполезен. Граф алгоритма приходится находить либо по текстам программ, либо из математических соотношений. Подобные формы записи почти всегда содержат параметры, например, размеры массивов, точность и т.п., которые на момент исследования записей не определены. Следовательно, вся работа по нахождению графа алгоритма и его исследованию неизбежно должна сводиться к решению параметризованных задач. Это очень серьёзная трудность. Тем не менее она была успешно преодолена.

В настоящее время решены различные связанные с реструктуризацией теоретические вопросы. Разработаны различные методы обнаружения параллельных ветвей вычислений. Предложены способы минимизации коммуникационных затрат при передачах информации между процессорами, а также между процессорами и памятью. Последнее имеет особое значение в связи с развитием распределенных вычислений. Сделано многое другое. Всё это сформировало новую область исследований, называемую *информационной структурой алгоритмов*. В её основе лежит выделение из записей алгоритма его информационного ядра, очищенного от всех элементов описания. Доказаны очень важные утверждения, из которых следует, что для широкого класса алгоритмов информационное ядро может быть описано и исследовано с помощью конечных наборов простых функций, как правило, кусочно-линейных. Построены эффективные методы вычисления, исследования и использования таких функций. Для изучения структуры алгоритмов создана и успешно функционирует автономная программная

система V-Ray. Она позволяет исследовать информационную структуру алгоритмов, описанных на языках Фортран и Си, и адаптировать программы под требования компьютеров параллельной архитектуры. Ясно, что параллельная структура программ является составной частью информационной структуры алгоритмов и может быть исследована с учетом всего сказанного выше.

Образование. Теперь можно попытаться ответить на вопрос, почему освоение вычислительной техники параллельной архитектуры идет с большими трудностями. На наш взгляд, это связано с тем, что знакомство с параллельными вычислениями, как и образование в этой области в целом, начинается не с того, с чего надо бы начинать. К тому же то, с чего надо начинать, не рассказывается ни в каких курсах вообще.

Как правило, знакомство с параллельными вычислениями начинается с изучения суперкомпьютеров и вычислительных систем параллельной архитектуры, языков и систем параллельного программирования. Безусловно, все это надо знать, и знать основательно. Однако и техника, и языки являются всего лишь *инструментами* для решения целевых задач пользователя. А *объектами*, обрабатываемыми с помощью таких инструментов, оказываются алгоритмы, с помощью которых задачи решаются. Из общих соображений ясно, что для проведения качественной обработки объекты и инструменты должны быть хорошо согласованы. Но как раз в осуществлении этого согласования и кроются многие трудности.

Для обычного пользователя инструментарий, т.е. техника и программное окружение, всегда является чем-то заданным. Если инструментарий и может быть изменен, то в очень ограниченных пределах. Остается подстраивать под его требования объекты, т.е. алгоритмы. Для этого необходимо обнаружить и описать параллельные ветви вычислений, разрезать алгоритм на фрагменты, минимизировать коммуникационные затраты, перестроить массивы данных. В зависимости от конкретных особенностей имеющейся вычислительной системы, возможно, придется делать и многое другое. Хорошо, если пользователь детально знает и задачу, и используемые алгоритмы, и особенности системы. В этом случае перестройка алгоритмов может и не вызвать серьезных затруднений. Но это относительно редкая ситуация. Чаще всего большие трудности приходится преодолевать и специалистам высокой квалификации. Что же тогда говорить о тех, кто только вступает в область параллельных вычислений!

В настоящее время основные вычислительные мощности достигаются на системах с общим числом отдельных процессоров, исчисляемых сотнями, тысячами и даже десятками тысяч. Эффективно решать задачи на таких системах трудно любым специалистам. Технологии и языки программирования, широко используемые на последовательных компьютерах и суперкомпьютерах с малым числом процессоров, на больших системах могут применяться лишь частично. Особенно трудно переносить большие

программные комплексы, создававшиеся разными людьми в течение долгого времени, на многопроцессорные кластеры и большие распределенные системы. Чтобы ни говорили сейчас специалисты по вычислительной технике и программированию, не видно никаких веских аргументов для предположений, что в обозримом будущем освоение новой вычислительной техники станет существенно проще. Поэтому к встрече с этим будущим надо серьезно готовиться уже сегодня.

Наиболее остро проблемы освоения новой техники стоят в связи с подготовкой молодых специалистов. Начать изменять создавшееся положение можно, например, корректируя те курсы, в которых будущий специалист впервые начинает серьезно изучать алгоритмы. Скорее всего, это курсы, касающиеся описания численных методов. Необходимо как можно раньше вводить в них первые элементы параллельных вычислений, например, в рамках упоминавшейся выше концепции неограниченного параллелизма. При описании численных методов совсем не трудно построить графы алгоритмов, указать в них параллельные ветви вычислений, ориентированные разрезы, которые определяют коммуникации, и многое другое, с чем придется столкнуться, работая с вычислительной техникой параллельной архитектуры.

Основная цель подобных изменений заключается в том, чтобы на самом раннем этапе обучения вычислительному делу вызвать интерес к информационным структурам алгоритмов и показать перспективность работы с ними. А перспективность действительно имеется.

Построение графов для большого числа конкретных алгоритмов выявило удивительную закономерность: большое разнообразие существующих методов не приводит к такому же разнообразию их информационных структур. Точнее, многие графы формально совершенно различных алгоритмов оказались изоморфными, отличаясь друг от друга только содержанием вершин и дуг. Поэтому была выдвинута гипотеза о том, что в конкретных вычислительных областях *типовых информационных структур немного*. Пока практика подтверждает эту гипотезу. Например, на всем множестве алгоритмов линейной алгебры типовых информационных структур оказалось всего лишь порядка десятка.

Специалистам по вычислительной и прикладной математике нужно сделать еще очень много, чтобы понять, как же в действительности устроены используемые ими алгоритмы. Если выдвинутая гипотеза окажется верной, то откроется много новых связей и направлений исследований. Изложение численных методов может быть поставлено на общий информационный фундамент, распараллеливание типовых информационных структур может быть заранее изучено и реализовано с помощью специальных программных средств, по типовым структурам могут быть построены спецпроцессоры, реализующие быстрое решение нужных алгоритмов. Отсюда уже недалеко и до построения заказных вычислительных систем,

ориентированных на эффективное решение классов задач из конкретных прикладных областей. В любом случае за выдвинутой гипотезой тянется много интересных следствий. Насколько гипотеза верна в реальности, покажет будущее. Но что-то интересное в ней всё-таки есть.

Возможно, следует более критически осмыслить научный фундамент параллельных вычислений. Мы уже отмечали значение машины Тьюринга для теории и практики последовательных вычислений. Этот формальный автомат позволяет реализовать любой последовательный алгоритм и в определённом смысле является прообразом любой однопроцессорной ЭВМ. Но любая многопроцессорная ЭВМ также может реализовать если не любой, то почти любой алгоритм. Тем не менее в развитии таких ЭВМ и всего того, что их окружает, совсем не ощущается необходимость введения формального автомата, который позволял бы реализовать *любой параллельный алгоритм*. Однако появилась острая потребность в некоторой идеализированной машине, на которой можно было бы реализовать для конкретного алгоритма *любой режим*, как параллельный, так и последовательный. С подобными задачами вполне справилась *граф-машина*, построенная на основе графа алгоритма.

Вообще говоря, не очень понятно, почему не появляется необходимость введения идеализированной параллельной машины, на которой можно было бы реализовывать любые алгоритмы в любом режиме. Может быть, просто потому, что развитие параллельной вычислительной техники до сих пор осуществляется в значительной мере стихийно и пока ещё не сформировались устойчивые принципы её конструирования?

Конечно, многое из сказанного ориентировано на перспективу. Однако использование различных знаний, касающихся информационной структуры алгоритмов, уже сейчас приносит свои плоды. Например, решена проблема построения математических моделей систолических массивов, представляющих специального вида вычислительные системы для сверхбыстрой реализации некоторых алгоритмов. Установлена связь графа алгоритма и таких задач, как быстрое вычисление производной и градиента, быстрое восстановление линейного функционала, оценивание влияния ошибок округления. Хотя граф алгоритма и является важнейшим понятием, связанным с изучением параллельных свойств алгоритмов, тем не менее, ни этот граф, ни все перечисленные только что задачи не имеют к параллелизму никакого прямого отношения. Может быть, между различными алгоритмическими проблемами имеется куда больше связей, чем известно сегодня?

Складывается впечатление, что значение сведений об информационной структуре алгоритмов выходит далеко за рамки параллельных вычислений и, возможно, эти сведения являются стержнем многих исследований в области алгоритмов, реализуемых на вычислительной технике. Если это

так, то первичные знания из данной области должны внедряться в образование. И чем скорее и на более ранних этапах, тем лучше.

Литература

1. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. − СПб:. БХВ-Петербург (2-е изд. 2004), 2002. − 608 с.

СОДЕРЖАНИЕ

Малышкин В.Э. Структурный синтез параллельных программ ……..…3
Старченко А.В. Параллельные вычисления в задачах исследования и прогноза погоды……..10
Копысов С.П., Пономарев А.Б., Рычков В.Н., Зубцовский С.Н. Расчетные неструктурированные сетки для распределенных вычислений……...19
Величко А.С. Параллельный прямо-двойственный алгоритм отсечений для решения структурированных линейных оптимизационных задач….....26
Берцун В.Н. Параллельный алгоритм для сплайновой системы обыкновенных дифференциальных уравнений……..35
Миньков Л.Л., Озерская К.В. Параллельная реализация явной разностной схемы для расчёта двухфазных течений в каналах……41
Малышенко В.В. Параллельная реализация метода частиц для решения задач гидродинамики со свободными границами…….50
Тимошевская Н.Е. О методах разработки параллельных комбинаторных алгоритмов…….60
Семенов В. В. Алгоритмы параллельного перечисления разбиений множества……..73
Беликов Д.А. Методы параллельных вычислений в задачах мониторинга качества атмосферного воздуха……...77
Барановский Н.В. Прогнозирование количества лесных пожаров и Ландшафтное распараллеливание……86
Данилкин Е.А., Старченко А.В. К выбору способа декомпозиции при численном решении систем связанных дифференциальных уравнений на многопроцессорной технике с распределённой памятью……..95
Сущенко М.С., Сущенко С.П. Проблемы организации многоуровневой памяти вычислительных систем……102
Легалов А.И. Использование асинхронно поступающих данных в потоковой модели вычислений……..113
Редькин А.В. Промежуточное представление функционального языка потокового программирования……..121
Гнедов Е.А. Модель вычислительных ресурсов с неоднородной коммутацией…….……125
Воеводин В.В. Математические проблемы параллельных вычислений…132