УДК 681.513.8

# ПРИНЦИПЫ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ В РЕЛАКСАЦИОННОМ ИТЕРАЦИОННОМ АЛГОРИТМЕ МГУА

А.В. Павлов

*Международный научно-учебный центр ЮНЕСКО информационных технологий и систем НАНУ и МОНУ, 03680, Киев, пр. Глушкова 40*

*andriypavlove@gmail.com*

У роботі запропоновані принципи і схеми паралельних обчислень в узагальненому релаксаційному ітераційному алгоритмі МГУА. Розроблені схеми дозволяють збільшити швидкість роботи алгоритму пропорційно кількості обчислючачів, досягаючи при цьому максимальної їх завантаженості.

*Ключові слова*: *узагальнений релаксаційний ітераційний алгоритм, паралельні обчислення, принципи, схеми, метод групового урахування аргументів.*

The paper suggests schemes and formulates principles of parallel computations for generalized relaxational iterative algorithm. The schemes developed allow to speedup the algorithm proportianaly to number of CPU cores with maximal load each of them.

*Key words*: *generalized relaxational iterative algorithm, parallel computations, principles, schemes, group method of data handling.*

В работе предложены принципы и схемы паралелльных вычислений в обобщённом релаксационном итерационном алгоритме МГУА. Разработанные схемы позволяют увеличить скорость работы алгоритма пропорционально количеству вычислителей, достигая при этом максимальной загрузке вычислителей.

*Ключевые слова: обобщённый релаксационный итерационный алгоритм МГУА, параллельные вычисления, принципы, схемы, метод группового учёта аугментов.*

## Вступление

Безостановочный технический прогресс в области вычислительных ресурсов побуждает инженеров, научных исследователей и программистов ставить перед собой новые научно-технические задачи, в первую очередь, декомпозицию и распределение вычислительно-трудоёмких задач на задачи меньшей сложности, которые могут быть решены параллельно на нескольких вычислителях за приемлемое время.

Сложность вычислений в алгоритмах метода группового учёта аргументов (МГУА) характеризуются двумя параметрами: $n$ – количество наблюдений, $m$ – число переменных. При этом, как отмечается в [1], число $m$ – может быть включать как исходные так и преобразованные переменные, что зависит от конкретного алгоритма МГУА. Хорошо известно, что алгоритмы МГУА имеют высокую сложность вычислений, в частности, итерационные – не менее порядка $nm^2 + m^3$[2] при осуществлении $m$ итераций и свободой выбора решений на каждой из них $F = m$, а комбинаторные – не менее порядка $nm^2 + m^4$ [3]. Однако, они достаточно легко могут быть распараллелены. Сделаем обзор существую-

щих решений по распараллеливанию как комбинаторных, так и итерационных алгоритмов МГУА.

## 1. Обзор работ

В первую очередь были распараллелены комбинаторные алгоритмы [4 5], поскольку являются более вычислительно трудоёмкими нежели итерационные, а также страдают «проклятием размерности» из-за полного перебора, что есть NP-сложная задача. Если однопроцессорные системы позволяют решать задачу комбинаторного перебора с количеством переменных $m$ = 20 за 10 сек., строя при этом $2^{20}$ = 1,048,576 моделей, то распараллеливание позволяет увеличивать число моделей пропорционально увеличению количества вычислителей системы, т.е., например, строить $2^{26}$ = 67,108,864 моделей за то же время, на 64 процессорах. Однако, всё же это не преодолевает NP-сложность этой задачи, поскольку, как показано в примере выше, это позволило увеличить число $m$ только на 6 переменных.

Основной задачей распараллеливания комбинаторных алгоритмов являлось равномерная загрузка всех процессоров, поскольку в зависимости от количества аргументов модели время её построения существенно различается [4].

В [4] предложено использовать обратные структурные вектора, т.е. процессор строит модель, структурный вектор которой имеет вид (1101110), он также обязан построить модель вида (0010001). Этот способ позволил получить загруженность процессоров на уровне 96% каким бы ни было число $m$ и количество процессоров. В [5] предложено использовать способ генерации структурных векторов с последовательным усложнением – изменением «0» на «1» в какой-либо ячейке вектора. Что позволило достичь почти максимальной загрузки – 99.8%.

Способы и принципы распараллеливания итерационных алгоритмов МГУА рассмотрены в работах [6-7].

В [6] предложено распараллеливание алгоритма GAME – нейронной МГУА-сети с различными типами нейронов и межслойными связями на базе симметричной многопроцессорной (SMP) архитектуры, позволяющей нескольким процессорам или ядрам обращаться к одной общей памяти. Поскольку каждый нейрон вычислительно независим от других, а время создание потоков несравнимо мало с временем расчёта одного нейрона, было принято решение рассчитывать каждый нейрон в отдельном параллельном потоке. Синхронизации требовал этап отбора нейронов, переходящих на следующий слой. Свойство расширяемости (scalability) GAME оказалось достаточно низкой: если для двух ядер получено ускорение в 1.7 раза, то для 8 ядер алгоритм работал только в 3.5 раз быстрей, чем последовательная версия.

Работа [7] рассматривает три разных вида ускорения: 1) за счёт распараллеливания задач по потокам (процессорам); 2) векторная параллельная обработка (векторизация), за счёт использования расширенного набора команд процессора (SSE – Streaming SIMD Extensions); 3) за счёт использования 64-разрядной операционной системы. Принцип распараллеливания задач был сле-

дующий: низкоуровневые базовые вычисления (простые циклы) выполняются используя векторный параллелизм, а более сложные (повторяющиеся функциональные задачи, состоящие из функций инициализации, манипуляции данными, множество вычислительных подзадач и циклов, чтение и запись данных) выполняются одновременно во многих потоках. Результаты расширяемости технологии распараллеливания по потокам показали почти линейных характер: начиная от 2 ядер – ускорение в 2 раза, заканчивая 8 ядрами – ускорение в 7.4 раза. Эти результаты можно обобщить на разрядность операционных систем и использование векторизации. Использование 64-разрядной системы даёт ускорение в 1.5 раза по сравнению с 32-х разрядной. Использование векторизации даёт ускорение в 1.5 раза.

## 2. Постановка задачи

В [1] описан наиболее быстрый на сегодняшний день итерационный алгоритм МГУА. В отличие от всех остальных алгоритмов на этапе построения моделей он не зависит от количества наблюдений и имеет линейную сложность относительно числа аргументов модели. Несмотря на его колоссальное быстродействие – построение модели от 30-ти аргументов (итераций), $n$ = 100000, $m$ = 1000 лишь за 2 мин. – это время можно уменьшить за счёт распараллеливание вычислений. Поэтому целью данной работы является выработка принципов и схем распараллеливания основных операций обобщённого релаксационного алгоритма (ОРИА).

## 3. Принципы распараллеливания операций в ОРИА

В работе предлагается осуществить распараллеливание операций на базе архитектуры SMP [7], рассмотренной в [6]. При распараллеливании нужно выделить участки алгоритма, выполнение которых занимает основное время его работы. Процесс работы ОРИА состоит из трёх стадий [1]: 1) преобразование исходной матрицы; 2) расчёт матриц нормальных уравнений; 3) стадия итераций, где осуществляется непосредственное построение моделей. Практически основное время работы алгоритма расходуется на последние две стадии. В зависимости от количества наблюдений $n$, числа аргументов $m$ и свободы выбора $F$, распределение времени между ними может быть любым. Поэтому распараллеливанию подлежат последние две стадии.

### 3.1 Распараллеливание стадии подготовительных вычислений

На стадии подготовительных вычислений необходимо рассчитать величины

$$\mathbf{X}_A^T\mathbf{X}_A,\ \mathbf{X}_B^T\mathbf{X}_B,\ \mathbf{X}_A^T\mathbf{y}_A,\ \mathbf{X}_B^T\mathbf{y}_B,\ \mathbf{y}_A^T\mathbf{y}_A,\ \mathbf{y}_B^T\mathbf{y}_B, \qquad (1)$$

где $\mathbf{X}=\begin{pmatrix}\mathbf{X}_A\\ \mathbf{X}_B\end{pmatrix}$, $\mathbf{y}=\begin{pmatrix}\mathbf{y}_A\\ \mathbf{y}_B\end{pmatrix}$ – матрица входных перменных и вектор выхода соответсвенно; индексы $A$ и $B$ обозначают обучающую и проверочную последова-

тельности соответственно. Алгоритм расчёта первых четырёх величин в (1) представлен ниже.

```
for ( i = 0; i < n; i++ )
  for ( j = i; j ≤ n; j++ )
  {
    if ( j ≠ n )
    {
      for ( l = 0; j < n_A; l++ )
      {
        value += X_A[i][l]*X_A[j][l];
      }
      X_A^T X_A[i][j] = value; value = 0;
      for ( l = 0; j < n_B; l++ )
      {
        value += X_B[i][l]*X_B[j][l];
      }
      X_B^T X_B[i][j] = value;
    }
    else
    {
      for ( l = 0; j < n_A; l++ )
      {
        X_A^T y_A[ind] += X_A[i][l]*y_A[l];
      }
      for ( l = 0; j < n_B; l++ )
      {
        X_B^T y_B[ind] += X_B[i][l]*y_B[l];
      }
      ind++;
    }
  }
```

Для равномерной загрузки вычислителей минимальная задача, подаваемая на вычислитель будет состоять из расчёта (1) для двух значений $i$, равноудалённых от значения $n/2$, т.е., для $i = 0$ и $i = n - 1$, $i = 1$ и $i = n - 2$,… Т.е. в общей сложности имеем $n/2$ задач при чётном $n$. Если n нечётно, $n/2 + 1$ задач, одна из которых средней сложности для значения $i = n/2 + 1$. Не ограничивая общности, пусть $n$ – чётное. Имея $k$ вычислителей, каждый из них получает:

$$N = \begin{cases} N_1 = n/(2k) \text{ задач, если } N_1 - \text{целое} \\ \text{первые } \lfloor n/(2k) \rfloor \text{ вычислителей получают } \lceil n/(2k) \rceil + 1, \text{ остальные } - \lceil n/(2k) \rceil \end{cases},$$

где $\lfloor\ \rfloor$ - остаток от деления, $\lceil\ \rceil$ - результат деления с отбрасываниям остатка. Расчёт величин $\mathbf{y}_A^T \mathbf{y}_A$, $\mathbf{y}_B^T \mathbf{y}_B$ можно реализовать паралельно на двух вычислителях.

### 3.2 Распараллеливание стадии построения моделей

Процесс построения модели

$$y = \sum_{i=1}^{m} \theta_i x_i$$

в ОРИА состоит в итеративном добавлении к текущей модели $\hat{y}_r$ $r$-й итерации нового аргумента $x_i \in \aleph = \{x_i\}_{i=\overline{1,m}}$, при этом строиться $m$ моделей вида

$$\hat{y}_{r+1,i} = \hat{\varpi}_{r,i}\hat{y}_r + \hat{\omega}_{r,i}x_{r,i},\ i \in I\ , \qquad (2)$$

где $I$ – множество индексов аргументов. На каждой итерации выполняется селекция $F$ лучших моделей $\{\hat{y}_{r+1,j}\}_{j=\overline{1,F}}$, которые будут усложняться по формуле (2) на следующей итерации ОРИА. Поэтому общее число моделей, которые нужно построить на $r$-й итерации равно $mF$.

Для поиска оценок коэффицентов $\hat{\varpi}_{r,i}$, $\hat{\omega}_{r,i}$ решается $Fm$ задач вида:

$$\{\hat{\varpi}_{r,i,j}, \hat{\omega}_{r,i,j}\} = \arg \min_{\varpi_{i,j}\in\Re, \omega_{i,j}\in\Re} |\mathbf{y}_A - \varpi_r \hat{\mathbf{y}}_{A,r,j} - \omega_r \mathbf{x}_{A,r,i}|\ ,\ i = \overline{1,m}\ ,\ j = \overline{1,F}\ , \qquad (3)$$

Для выбора $F$ лучших моделей $r + 1$ итерации решаются следующие $Fm$ задач:

$$\{\hat{\mathbf{y}}_{B,r+1,l}\}_{l=\overline{1,F}} = \arg \min_{i=\overline{1,m}, j=\overline{1,F}} |\mathbf{y}_B - \hat{\mathbf{y}}_{B,r+1,i,j}| = \arg \min_{i=\overline{1,m}, j=\overline{1,F}} |\mathbf{y}_B - \hat{\varpi}_r \hat{\mathbf{y}}_{B,r,j} - \hat{\omega}_r \mathbf{x}_{B,r,i}| \qquad (4)$$

Таким образом на каждой итерации имеем $Fm$ задач (3), (4), которые могут решаться независимо.

Эта стадия распараллеливается аналогично работе [6], однако количество создаваемых потоков должно быть равно количеству вычислителей, дабы операционная система не тратила время на переключение между потоками, как это происходит в [6], когда количество потоков больше числа вычислителей. Синхронизации потоков подлежит стадия селекции лучших $F$ моделей.

Пусть мы имеем $k$ вычислителей, тогда при распределении $Fm$ задач могут возникнуть такие случаи:

1. $Fm < k$. Каждый из $Fm$ первых вычислителей загружен одной задачей, остальные $(k - Fm)$ – свободны.

2. $Fm = k$. Все вычислтели загружены и каждый из них решает одну задачу.

3. $Fm > k$. Первые $\lfloor Fm/k \rfloor$ вычислителей решают $\lceil Fm/k \rceil + 1$ задач, а остальные – $\lceil Fm/k \rceil$ задач.

Предложенные схемы распараллеливания двух основных стадий ОРИА позволяют получить максимальную загрузку вычислителей при любом их количестве.

## 4. Выводы

Предложенные в работе принципы и схемы распараллеливаня вычислений обобщённого релаксационного итерационного алгоритма МГУА позволяют увеличить скорость работы алгоритма пропорционально количеству вычислителей при максимальной (почти равномерной) их загрузке.

## Литература

1. Павлов А. В. Обобщённый релаксационный итерационный алгоритм МГУА // Індуктивне моделювання складних систем. Зб. наук. праць, вип. 2. – К.: МННЦІТС НАНУ, 2011. – С. 95-108.

2. А. В. Павлов, В. С. Степашко Рекуррентные алгоритмы расчёта коэффициентов и критериев селекции в релаксационном алгоритме МГУА // Кибернетика и вычислительная техника. – 2011. – Вып. 165. – С. 72-82.

3. Єфіменко С.М., Степашко В.С. Рекурентний алгоритм методу Гауса для розв'язання систем лінійних рівнянь у задачі оцінювання параметрів регресійних моделей // Відбір і обробка інформації. Міжвідомчий збірник наукових праць. – №36 (112). – 2012. – С. 48-55

4. O.A. Koshulko, A.I. Koshulko Adaptive parallel implementation of the Combinatorial GMDH algorithm. // Proc. of IWIM 2007. International workshop on inductive modeling, CTU in Prague, 2007, p. 71-77. ISBN 978-80-01-03881-9

5. Volodymyr Stepashko, Serhiy Yefimenko Optimal Paralleling for Solving Combinatorial Modelling Problems // Proc. of ICIM 2008. 2nd International conference on inductive modeling, Kyiv, 2008, p. 172-175. ISBN 978-966-02-4889

6. Pavel Kordik, Jakub Spirk, Ivan Simecek Parallel computing of GAME models // Proc. of ICIM 2008. 2nd International conference on inductive modeling, Kyiv, 2008, p. 160-163. ISBN 978-966-02-4889

7. Frank Lemke Parallel Self-organizing Modeling // Proc. of ICIM 2008. 2nd International conference on inductive modeling, Kyiv, 2008, p. 176-183. ISBN 978-966-02-4889

8. http://en.wikipedia.org/wiki/Symmetric_multiprocessing.