ISSN 1512-1232

УДК 519.684.4

ОПТИМИЗАЦИЯ ПАРАЛЛЕЛЬНОГО АЛГОРИТМА СИНТЕЗА МОДЕЛИ ОПТИМАЛЬНОЙ СЛОЖНОСТИ НА ПРИНЦИПАХ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ

Михаил Иванович Горбийчук, Богдан Васильевич Пашковский [1]
[1] Ивано-Франковский национальный технический университет нефти и газа,
ул. Карпатская 15, г. Ивано-Франковск, Украина, 76019

Для сложных технических объектов с большим числом входных переменных затраты машинного времени на построение математической модели достаточно заметны. Анализ метода построения эмпирических моделей оптимальной сложности на основе генетических алгоритмов показал, что такой алгоритм имеет внутренний параллелизм, что позволяет разработать эффективную программу реализации, что приведет к сокращению затрат машинного времени.

Наиболее затратными операциями такого алгоритма является решение системы линейных алгебраических уравнений. Эти операции выполняются многократно, поэтому для уменьшения затрат машинного времени были разработаны параллельные алгоритмы их реализации.

Существующие алгоритмы при оценке своей быстродействия учитывают только время используемое при решении системы линейных алгебраических уравнений и не учитывают время передачи или считывания данных, которое может быть существенно больше, поэтому такая оценка не может служить адекватным критерием быстродействия алгоритма.

Проанализированы быстродействия существующих алгоритмов параллельного решения системы линейных алгебраических уравнений с учетом времени на передачу и считывания данных.

Реализован алгоритм, позволяющий существенно уменьшить затраты машинного времени на решение системы линейных алгебраических уравнений.

Ключевые слова: *эмпирическая модель, параллелизм, C ++ MPI, C #, NET, TPL.*

1. Введение

Для получения модели сложной системы, которая содержит много компонентов, которые сложным образом взаимодействуют между собой и описания взаимодействия между параметрами системы и внешней средой, что непосредственно действует на систему и вызывает изменение ее состояния, используют эмпирическое моделирование.

Полученные таким образом математические модели широко используются для решения таких задач как распознавание образов, прогнозирование, автоматическая классификация, оптимальное управление и другие [1].

Синтез таких моделей является сложным и требует значительных затрат машинного времени, поэтому целесообразно рассмотреть возможность параллельной реализации таких алгоритмов.

Обнаружен внутренний параллелизм алгоритма синтеза эмпирических моделей оптимальной сложности [2] позволил реализовать его программными средствами и определить его показатели - ускорение и эффективность [3].

Однако оценка быстродействия алгоритма учитывает только решение системы линейных алгебраических уравнений и не учитывает время передачи или считывания данных, что может иметь существенное влияние на время выполнения алгоритма в целом.

ISSN 1512-1232

Поэтому актуальным является исследование быстродействия имеющихся алгоритмов с учетом времени на передачу и считывания данных, рассмотрение различных технологий построения параллельных программ, и оптимизация алгоритма синтеза эмпирических моделей оптимальной сложности.

2. Анализ литературных данных

В основе эмпирического моделирования многих явлений и процессов лежит широко известный метод наименьших квадратов (МНК), основанный более 200 лет назад молодым немецким математиком Карлом Фридрихом Гауссом.

МНК предполагает, что известна структура модели и необходимо по наблюдениям как по входным, так и по выходным величинами построить модель, которая наилучшим образом аппроксимировала бы эмпирические данные. В качестве критерия приближения выбирают сумму квадратов отклонений экспериментальных данных от расчетных, полученных с эмпирической моделью.

Критерий приближения МНК является внутренним критерием [4] и его применение приводит к ошибочному выводу: чем сложнее модель, тем она точнее. Поскольку на выход системы накладывается помеха (допускают, что она аддитивная), то чрезмерная точность модели может значительно исказить объективно существующую функциональную зависимость между выходом системы и ее входами.

Поэтому для выбора структуры эмпирической модели акад. А. Г. Ивахненком был предложен метод, получивший название индуктивный метод самоорганизации моделей [5], идейную сторону которого определяет теорема Геделя. Эта теорема утверждает, что никакая система аксиом не может быть логически замкнутой: всегда найдется такая теорема, для доказательства которой необходимо внешнее дополнение - расширение исходной системы аксиом. Относительно задачи синтеза модели геделевский подход означает применение внешнего критерия для однозначного выбора структуры модели из заданного класса моделей. Критерий называют внешним, если его вычисления основывается на данных, которые не использовались при решении задачи МНК.

Задача синтеза эмпирических моделей с использованием индуктивного метода самоорганизации моделей относится к классу задач большой размерности. Например, если выбрана полиномиальная модель, которая имеет 7 аргументов и пятую степень полинома, то общее число моделей-претендентов, среди которых следует выбрать модель оптимальной сложности составит [6] $\approx 2{,}6\cdot 10^{238}$. Понятно, что реализовать такой объем вычислений даже на самых новых ЭВМ невозможно.

Для снижения размерности задачи авторами работы [6] предложен метод синтеза эмпирических моделей на основе генетических алгоритмов, суть которого будет представлена ниже. Это позволило реализовать соответствующий алгоритм в котором число входных переменных не превышает семи, а степень полинома не более пяти. При этом затраты машинного времени удалось радикально уменьшить, но они остаются еще достаточно заметными. Одним из эффективных путей уменьшения таких затрат является применение параллельных вычислений, в которых на сегодняшний день привлечено значительное внимание как отечественных, так и зарубежных исследователей [1, 7-9].

3. Цель и задачи исследования

Целью исследования является оптимизация алгоритма синтеза моделей оптимальной сложности на основе генетических алгоритмов, за счет уменьшения машинного времени на считывание и передачу данных.

Задачами исследования являются:

- Анализ быстродействия существующих методов синтеза моделей оптимальной сложности на основе генетических алгоритмов, с учетом времени на чтение/передачу данных;

ISSN 1512-1232

- Анализ существующих технологий распараллеливания данных и их быстродействия.
- Уменьшение времени работы алгоритма за счет уменьшения машинного времени на считывание и передачу данных и, возможно, изменения технологии распараллеливания.

4. Исследование алгоритма решения системы линейных алгебраических уравнений на наличие внутреннего параллелизма

Исследуем на параллелизм алгоритмы решения системы линейных алгебраических уравнений (1).

$$\tilde{A}\bar{a}_A = \bar{b}\,, \tag{1}$$

Решение системы (1) осуществляется в два этапа: первый приведения матрицы $\tilde{A}$ к верхнему диагональному виду с единицами на главной диагонали с одновременным пересчетом свободных членов системы уравнений; второй этап нахождения решений системы уравнений (1) методом обратного хода.

Создадим расширенную матрицу $A_b = \left[\tilde{A} \mid \bar{b}\right]$ размером $(M-c)\times(M-c+1)$ и приведем ее к верхнему диагонального вида, осуществив такие операции над ее элементами:

$$a_{kj}^{(k)} = \frac{a_{kj}^{(k-1)}}{a_{kk}^{(k-1)}},\ a_{ij}^{(k)} = a_{ij}^{(k-1)} - a_{ik}^{(k-1)}a_{kj}^{(k)},\ i = \overline{k+1,z},\ j = \overline{k+1,z+1},\ k = \overline{1,z}\,, \tag{2}$$

где $z = M - c$.

Результатом преобразования матрицы A_b по формулам (14) будет верхняя диагональная матрица

$$U = \begin{bmatrix} 1 & u_{12} & u_{13} & \cdots & u_{1z} & u_{1,z+1} \\ 0 & 1 & u_{23} & \cdots & u_{2z} & u_{2,z+1} \\ 0 & 0 & 1 & \cdots & u_{3z} & u_{3,z+1} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & 1 & u_{z,z+1} \end{bmatrix}$$

Обратный ход, в котором в обратном порядке определяются неизвестные осуществляется по формулам

$$a_{A,z} = u_{z,z+1},\ a_{A,i} = u_{i,z+1} - \sum_{j=i+1}^{z} u_{ij}a_{A,j},\ i = \overline{z-1,1}. \tag{3}$$

Построим граф алгоритма (2). Вершины графа соответствуют операциям типа a/b и $a - cd$. Для того, чтобы облегчить анализ графа алгоритма, вершины графа разместим в определенном порядке. Рассмотрим прямоугольную решетку в трехмерном пространстве с координатами i, j, k Во все целочисленные узлы решетки для $j = \overline{1,z+1},\ i,k = \overline{1,z}$ поместим вершины графа. Анализируя формулы (2), можно прийти к выводу, что в вершину с координатами i, j, k будет передаваться результат выполнения операции, который соответствует вершине с координатами $i, j, k-1$. Для $k = 2$ граф алгоритма показан на рис. 2. Параллелизм алгоритма (2) проявляется в том, что для каждого k пересчет элементов матрицы A_b осуществляется независимо.

ISSN 1512-1232

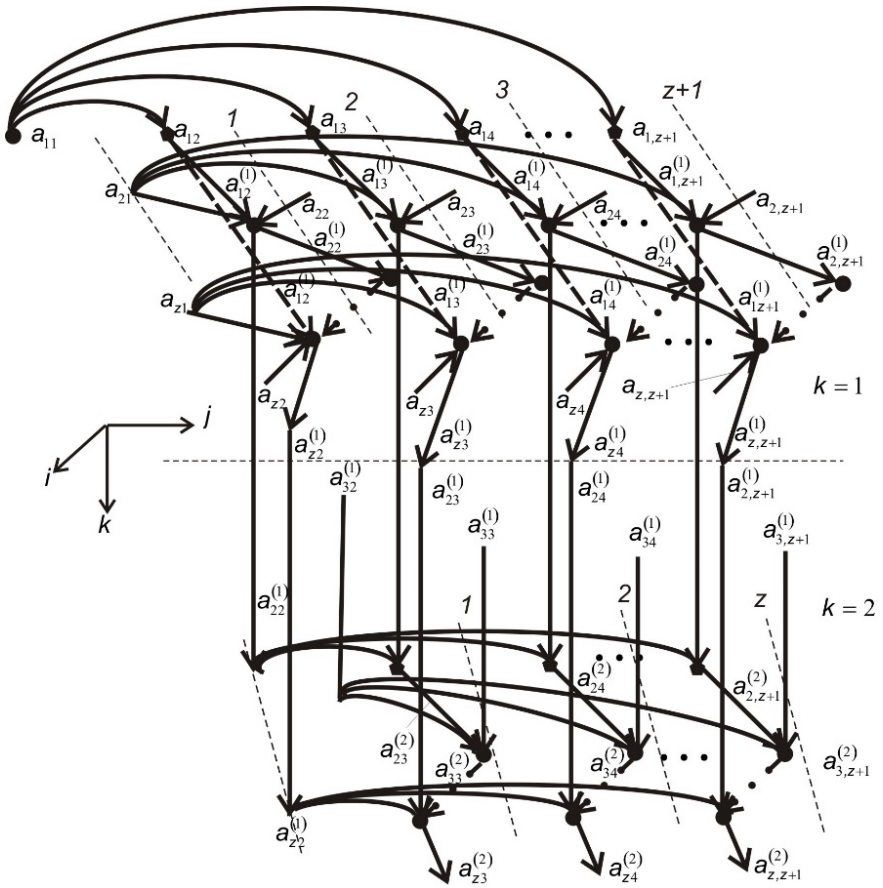

Рис. 1. Граф алгоритма приведения матрицы A_b к верхней треугольной форме

5. Построение параллельного алгоритма решения методом Гаусса.

Поскольку решение методом Гаусса сводится к последовательности однотипных вычислительных операций умножения и сложения над строками матрицы, как подзадачу можно принять вычисления, связанные с обработкой одной или нескольких строк матрицы A и соответствующего элемента вектора b. Каждая итерация, связанная с решением очередной подзадачи, начинается с выбора ведущего строки. Ищется строка с наибольшим по абсолютной величине значением среди элементов i столбца, соответствующего переменной x_i, что исключается.

Поскольку строки матрицы A закреплены за различными подзадачами, для поиска максимального значения в столбце подзадачи с номерами k, должны обменяться элементами при переменной x_i, которая исключается. После сбора всех указанных коэффициентов может быть определено, какая из подзадач содержит ведущий строку и какое значение является ведущим элементом.

Для продолжения вычислений ведущая подзадача должна разослать свою строку матрицы A и соответствующий элемент вектора b всем другим подзадачи с номерами k, k_{xi}.

Как подзадачи могут быть взяты строки матрицы A, каждая из которых при этом закрепляется за одним процессором. Если число строк матрицы больше, чем число доступных процессоров (s, подзадачи можно укрупнить, объединив несколько строк матрицы)

$$T_p' = \sum_{i=0}^{n-2} \frac{n-i}{s}(2(n-i)+1)$$

ISSN 1512-1232

Анализ эффективности параллельных вычислений методом Гаусса

Пусть n - порядок системы линейных уравнений, а s - число используемых процессоров, то есть матрица A имеет размер n × n, а n/s - размер полосы на каждом процессоре (для простоты считаем, что n/s - целое число). Определим сложность параллельного варианта метода Гаусса [10].

В прямом ходе алгоритма для выбора ведущего строки на каждой итерации и каждом процессоре должно быть определено максимальное значение в столбце с неизвестной, что будет выключаться в пределах закрепленной за ним полосы. При изъятии неизвестных количество строк в полосах сокращается. После сбора полученных максимальных значений, определения и рассылки ведущей строки на каждом процессоре выполняется вычитание ведущего строки из каждой строки остальных строк своей полосы матрицы A.

Общие затраты на выполнение действий в одной строке на i-й итерации, составляет 2(n-i)+1 операций. Если применена циклическая схема распределения данных между процессорами, то на i-й итерации каждый процессор будет обрабатывать примерно (n - i) / s строк [10].

С учетом сказанного, общее число операций параллельного варианта прямого хода метода Гаусса определяется выражением

$$T_p' = \sum_{i=0}^{n-2} \frac{n-i}{s} \cdot (2(n-i)+1)$$

Заметим, что (n - i) / s, как правило, не будет целым. Для построения приближенных достаточных оценок не будем учитывать операцию округления, но на каждой итерации введем операции с одной дополнительным строкой:

$$T_p' = \sum_{i=0}^{n-2} \frac{n-i}{s} (2(n-i)+1) + \sum_{i=0}^{n-2} (2(n-i)+1)$$

$$T_p' = \frac{1}{s} \sum_{i=0}^{n-2} \left((n-i) + 2(n-i)^2\right) + \sum_{i=0}^{n-2} (2(n-i)+1)$$

На каждой i-й итерации обратного хода после рассылки очередной исчисленной неизвестной на каждом процессоре обновляются значения правых частей для (n - i) / s еще не обработанных строк из числа закрепленных за ним n / s строк. Поскольку для каждой правой части выполняются 2 операции (умножение и вычитание), общее число операций, необходимых для обновления всех правых частей, составит

$$T_p'' = \sum_{i=0}^{n-2} 2\frac{n-i}{s}$$

Кроме обновления правых частей на каждой итерации обратного хода один из процессоров выполняет операцию деления для определения очередной переменной. При этом другие процессоры, конечно, простаивают, но эти n операций (по числу определенных переменных) необходимо включить в общие вычислительные затраты [10]:

$$T_p'' = 2\sum_{i=0}^{n-2} \frac{n-i}{s} + n$$

Наконец, для построения приближенных достаточных оценок, отменим операцию округления и добавим 2(n – 1) операций, которые могут дополнительно иметь место на всех

ISSN 1512-1232

2(n – 1) итерациях обновление правых частей, если в результате округления на каждой прилагается одну строку. Тогда оценка сверху общих вычислительных затрат для реализации обратного хода составит

$$T_p'' = 2\sum_{i=0}^{n-2}\frac{n-i}{s} + n + 2(n-1) = \frac{1}{s}\sum_{i=0}^{n-2} 2(n-i) + 3n - 2$$

Таким образом суммарные затраты на реализацию прямого и обратного хода составят:

$$T_p = \frac{1}{s}\sum_{i=0}^{n-2}\left(2(n-i)^2 + 3(n-i)\right) + \sum_{i=0}^{n-2}(2(n-i)+1) + 3n - 2$$

Осуществим замену j = ni, при этом границы оператора суммы при записи слагаемых в обратном порядке будут j = 2.n. тогда

$$T_p = \frac{1}{s}\sum_{j=2}^{n}(2j^2 + 3j) + \sum_{j=2}^{n}(2j+1) + 3n - 2$$

Применяя элементарные формулы для подсчета сумм, получаем

$$T_p = \frac{1}{s}\left(\frac{4n^3 + 15n^2 + 11n - 30}{6}\right) + (n^2 + 5n - 5)$$

Если это допустимо, при больших n можно применять приближенную оценку:

$$T_p = \frac{1}{s}\left(\frac{2}{3}n^3 + \frac{15}{16}n^2\right) + (n^2 + 5n - 5)$$

Учтем теперь расходы на передачу данных. В прямом ходе на каждой итерации для определения ведущей строки процессоры обмениваются найденными в своей полосе максимальными значениями в столбце с переменной, исключается. Для этого необходимо log2 (s) шагов. С учетом количества итераций общие затраты на передачу максимальных значений

$$T_p'(comm) = (n-1)\log_2 s\,(\propto + \frac{\omega}{\beta})$$

где α - латентность сети передачи данных, β - пропускная способность сети, и ω - размер элемента пересылаемых данных.

На каждой итерации прямого хода метода Гаусса выполняется также рассылка избранное ведущей строки. Сложность данной операции передачи данных определяется величиной

$$T_p''(comm) = (n-1)\log_2 s\,(\propto + \frac{\omega}{\beta})$$

Наконец, при выполнении обратного хода алгоритма Гаусса на каждой итерации происходит рассылка всем процессорам значений очередной неизвестной. Общее время, что требуется на рассылку [10]:

$$T_p'''(comm) = (n-1)\log_2 s\,(\propto + \frac{\omega}{\beta})$$

Таким образом общая сложность параллельного алгоритма Гаусса составит

$$T_p = \frac{1}{s}\sum_{j=2}^{n}(2j^2 + 3j)\tau + (n-1)\tau + (n-1)\log_2 s\,(3\alpha + \omega(n+2)/\beta)$$

где τ - время выполнения одной вычислительной операции.

ISSN 1512-1232

С использованием приближенных оценок вычислительной сложности последовательного и параллельного алгоритмов можно построить также оценки ускорения и эффективности [10]:

$$R = \frac{(\frac{2}{3}n^3 + \frac{2}{3}n^2)\tau}{\frac{1}{s}\left(\frac{2}{3}n^3 + \frac{15}{16}n^2\right)\tau + (n^2 + 5n - 5)\tau + (n-1)\log_2 s\,(\frac{3\alpha + \omega(n+2)}{\beta})}$$

$$E_p = \frac{(\frac{2}{3}n^3 + \frac{2}{3}n^2)\tau}{\left(\frac{2}{3}n^3 + \frac{15}{16}n^2\right)\tau + s(n^2 + 5n - 5)\tau + s(n-1)\log_2 s\,(\frac{3\alpha + \omega(n+2)}{\beta})}$$

6. Оптимизация параллельного алгоритму решения систем линейных алгебраических уравнений.

Большинство известных алгоритмов параллельного программирования используют язык программирования C++ и библиотеку MPI (message passing interface), недостатком такой библиотеки является то, что каждый процесс вынужден работать с локальной копией данных, в результате чего тратится время на обмен данными между процессами.

В качестве альтернативы был разработан алгоритм решения систем линейных алгебраических уравнений с использованием языка C# и библиотеки TPL (task parallel library). Эта библиотека дает более полный контроль над параллельными потоками и позволяет проводить операции над совместными данным минимизирует затраты времени на передачу данных.

Так же можно использовать OpenMP – открытый стандарт для распараллеливания программ с общей памятью, исследования [11] показали, что OpenMP может быть в 2-3 раза быстрее MPI. Сравнение з OpenMP будет рассмотрено в следующих научных работах.

Сравнительный алгоритмов проводился на процессоре с 4 физическими ядрами. Входом алгоритма служили квадратные матрицы размером 800, 1200, 2400 и 3200.

Применяется циклическое распределение строк.

Результаты работы алгоритмов над матрицей 2 400 отражены в табл. 1.

Таблица 1. Результаты общего времени приведения квадратной матрицы размером 2400 к верхнему треугольному виду с учетом времени на считывание данных.

Время обработки матрицы, с. (матрица 2400 на 2400)		
Количество потоков	C++	C# (safe)
1	152.702	281.1610961
2	130.154	143.052187
4	124.549	78.9015185
8	135.846	78.5054954
16	183.508	80.5766094

ISSN 1512-1232

Построив графики зависимости времени выполнения от количества потоков (рис.2) видно, что при 4 и более потоках, время выполнения алгоритма на языке C# меньше чем время выполнения на языке C ++.

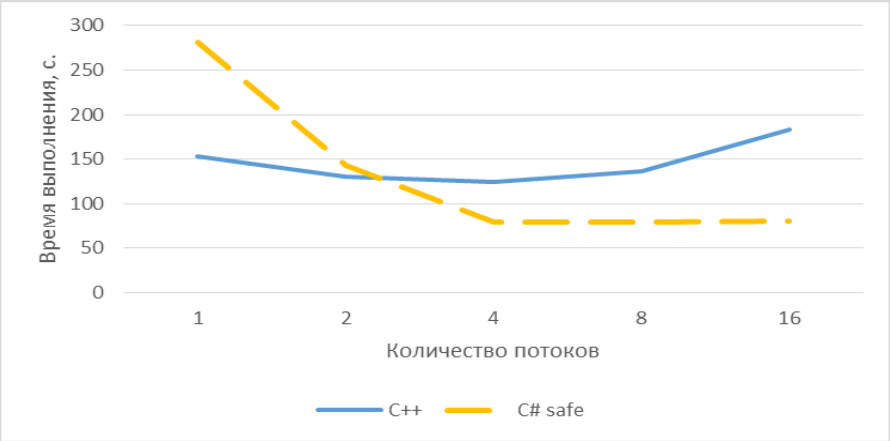

Рис. 2. Зависимость общего времени приведения квадратной матрицы размером 2400 к верхнему треугольному виду от количества потоков с учетом времени на считывание данных

Алгоритм состоит из двух основных частей - считывание входящей матрицы из файла и математических операций над матрицей в памяти. Поэтому целесообразно проанализировать время выполнения обеих составляющих алгоритмов.

Таблица 2. Результаты чтения квадратной матрицы размером 2400 из текстового файла

Время обработки матрицы, с. (матрица 2400 на 2400)		
Количество потоков	C++	C# (safe)
1	106.4702	1.7000923
2	106.0913	1.7000977
4	105.323	1.7021026
8	108.333	1.731099
16	106.2044	1.6830966

С таблицы 2 видно, что время считывания матрицы из файла с помощью языка C# почти на два порядка меньше, чем на языке C ++. Это связано со свойствами работы данных языков с файловой системой.

Таблица 3. Результаты приведения квадратной матрицы размером 2400 к верхнему треугольному виду

Время обработки матрицы, с. (матрица 2400 на 2400)		
Количество потоков	C++	C# (safe)
1	46.2318	279.4610038
2	24.0627	141.3520893
4	19.226	77.1994159
8	27.513	76.7743964
16	77.3036	78.8935128

ISSN 1512-1232

В таблице 3 приведены время приведения квадратной матрицы размером 2400 к верхнему треугольному виду без учета времени на считывание файла.

По данным таблицы построим графические зависимости (рис. 3).

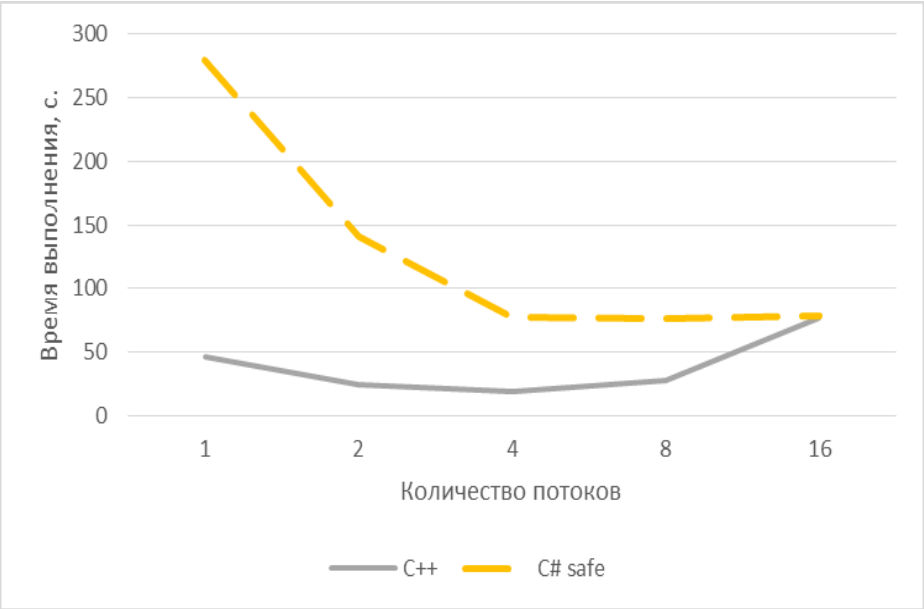

Рис. 3. Зависимость времени приведения квадратной матрицы размером 2400 к верхнему треугольному виду от количества потоков

Как видно из рис.3 математические операции в языке C # гораздо более ресурсоемкие, что связано с тем, что любое число представляется в виде объекта, и, как следствие требует гораздо большего использования оперативной памяти и процессорного времени на обработку.

Следует также отметить, что время выполнения на языке C++ при 8 и 16 потоках требует больше времени чем при 4, поскольку использовался компьютер с 4 физическими ядрами, и при большем количестве потоков не происходит распараллеливаниеи и тратятся ресурсы на коммуникацию между потоками.

Возникает вопрос, можно ли сократить время обработки на языке C # чисел, сохранив при этом существующие преимущества при работе с файловой системой, и тем самым существенно уменьшить время выполнения алгоритма в целом.

Язык C# позволяет писать так называемый неуправляемый (unsafe) код, который близок к коду C ++, и намного быстрее чем обычный управляемый (safe) код - который стандартно используется язык C#. Преимуществом такого подхода является выигрыш во времени выполнения, однако разработчику необходимо следить за высвобождением памяти, поскольку работа с памятью при таком программировании проходит практически в «ручном режиме».

Нами был разработан алгоритм приведения квадратной матрицы к верхней треугольного вида, с использованием языка C# и неуправляемого (unsafe) подхода.

Сравнительная характеристика времени приведения квадратной матрицы к верхней треугольной вида без учета времени на считывание данных приведена в таблице 4.

ISSN 1512-1232

Таблица 4. Результаты приведения квадратной матрицы размером 2400 до верхнего треугольного вида

Время обработки матрицы, с. (матрица 2400 на 2400)			
Количество потоков	C++	C# (safe)	C# (unsafe)
1	46.2318	279.4610038	49.9838567
2	24.0627	141.3520893	25.8354794
4	19.226	77.1994159	15.8939103
8	27.513	76.7743964	16.3689373
16	77.3036	78.8935128	15.7799034

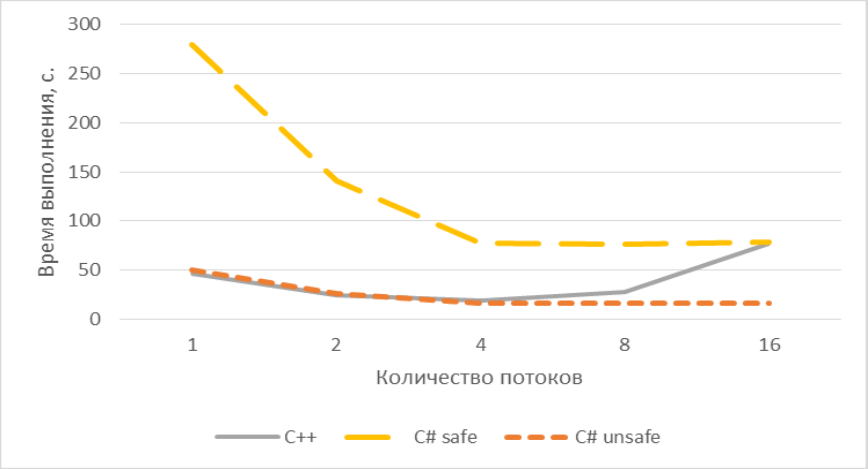

Рис. 4. Зависимость времени приведения квадратной матрицы размером 2400 к верхнему треугольному видуот количества потоков

Как видно из таблицы 4 и рисунка 4, время обработки время приведения квадратной матрицы к верхнему треугольному виду, с помощью неуправляемого подхода на языке C# намного меньше чем при использовании управляемого подхода и даже быстрее чем при использовании языка C ++.

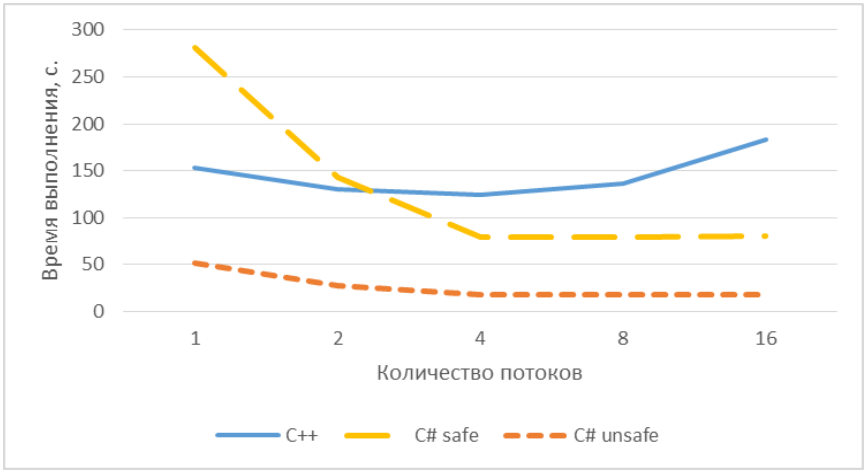

Рис. 5. Зависимость общего времени приведения квадратной матрицы размером 2400 к верхнему треугольному виду от количества потоков с учета времени на считывание данных

ISSN 1512-1232

Учитывая ощутимый выигрыш во времени считывания файла, общее время выполнения алгоритма с применением неуправляемого подхода на языке C # ощутимо меньше, чем при применении языка C ++, как в существующих алгоритмах.

Выводы

Анализ существующих алгоритмов решения систем линейных алгебраических уравнений показал большие затраты времени на считывание и передачу данных, связанных с использованием технологии C ++ / MPI.

Предложена другая технология распараллеливания данных C# / TPL, которая лучше оптимизирована для работы с файловой системой, и позволяет потокам работать с одними и теми же данными не создавая копий, уменьшает время работы такого алгоритма.

Анализ стандартного подхода к программированию такой технологии C # / TPL, так называемого управляемого (safe) кода показал значительные затраты времени на выполнение собственно математических операций.

Предложенный альтернативный метод, использование неуправляемого (unsafe) кода и разработана программная реализация такого метода.

Сравнение времени выполнения алгоритмов, показало, что применение неуправляемого подхода дает лучшие результаты при выполнении собственно математических операций над матрицами, в сочетании с выигрышем во времени при работе с файловой системой и уменьшением времени на обмен данными между потоками (за счет использования технологии C#/TPL) дает выигрыш во времени примерно в 7 раз, по сравнению с использованием технологии C ++ / MPI.

В статье содержится 5 рисунков и 4 таблицы

Литература

1. Воеводин В. В. Параллельные вычисления / В. В. Воеводин, Вл. В. Воеводин. – СПб: БХВ-Петербург, 2002. – 608 с.
2. Горбійчук, М. І. Паралелізм алгоритму синтезу моделей оптимальної складності на засадах генетичних алгоритмів [Текст] / М. І. Горбійчук, В. М. Медведчук, Б. В. Пашковський // Восточно-Европейский журнал передовых технологий – 2014. – № 4/2(70). – С. 42–48.
3. Горбійчук, М. І. Паралельний алгоритму синтезу моделей оптимальної складності на засадах генетичних алгоритмів [Текст] / М. І. Горбійчук, М.О.Слабінога, В. М. Медведчук // Методи та прилади контролю якості – 2013. – № 2(31). – С. 99–108.
4. Ермаков, С. М. Математическая теория оптимального эксперимента [Текст] / С. М. Ермаков, А. А. Жиглявский. – М.: Наука, 1987. – 320 с.
5. Ивахненко, А. Г. Индуктивный метод самоорганизации моделей сложных систем [Текст] / А. Г. Ивахненко. – К.: Наукова думка, 1981. – 296 с.
6. Горбійчук, М. І. Метод синтезу емпіричних моделей на засадах генетичних алгоритмів [Текст] / М. І. Горбійчук, О. Б. Василенко, І. В. Щупак // Розвідка та розробка нафтових і газових родовищ. – 2009. – № 4(33). – С. 72–79.
7. Ортега Дж. Введение в параллельные и векторные методы решения линейных систем / Ортега Дж.; пер. с англ. Х. Д. Икрамова, И. Е. Капорина под ред. Х. Д. Икрамова. – М.: Мир, 1991. – 367 с.
8. Бэбб Р. Программирование на параллельных вычислительных системах / Р. Бэбб, Дж.Мак-Гроу, Т. Акселрод и др.; пер. с англ. А. С. Косачева, Л. В. Шабанова под ред. Ю. Г. Дадаева. – М.: Мир, 1991, – 376 с.
9. Химич А. Н. Параллельные алгоритмы решения задач вычислительной математики / А. Н. Химич, И. Н. Молчанов, А. В. Попов и др. – К.: Наукова думка, 2008. – 248 с.

ISSN 1512-1232

10. Гергель, В.П., Стронгин, Р.Г. (2003, 2 изд.). Основы параллельных вычислений для многопроцессорных вычислительных систем. - Н.Новгород, ННГУ.
11. Sol Ji Kang, Sang Yeon Lee, and Keon Myung Lee. Performance Comparison of OpenMP, MPI, and MapReduce in Practical Problems. – Advances in Multimedia Volume 2015 (2015), Article ID 575687, 9 pages

Статья получена: 2015-11-02