электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

УДК 51.73

Оптимизация метода продольно-поперечной прогонки для решения уравнения теплопроводности

***Корепанов К.Е.**, магистр*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Программное обеспечение ЭВМ и информационные технологии»

***Строганов Ю.В.**, магистр*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Программное обеспечение ЭВМ и информационные технологии»

Научный руководитель: Градов В.М., д.т.н., профессор
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана
gradov@bmstu.ru

При моделировании различных физических процессов используются уравнения в частных производных. В настоящей работе рассматривается краевая задача стационарной двумерной теплопроводности с достаточно произвольной функцией источника. Уравнение решается методом установления [1]. При высокой требуемой точности вычисления могут занимать длительное время. Современные многоядерные процессоры позволяют использовать распараллеливание вычислений для ускорения выполнения программы.

В данной статье исследуется скоростные параметры продольно-поперечной разностной схемы для решения дифференциального уравнения теплопроводности при различном числе потоков. Метод переменных направлений детально описан в [1].

Время выполнения метода переменных направлений зависит от количества итераций и числа дискретных ячеек, на которые разбита область решения (прямоугольная пластина).

Распараллеливание алгоритма

При прогонке по одному из двух направлений – в области решения – строкам (столбцам) вычисление определенной строки не зависит от результатов вычисления предыдущей строки (столбца). Таким образом, допускается распараллеливание алгоритма для одновременного вычисления нескольких строк (столбцов). При этом не допускается

начало прогонки по столбцам до завершения прогонки по всем строкам. Общий алгоритм распараллеливания метода прогонки приведен на рис. 1.

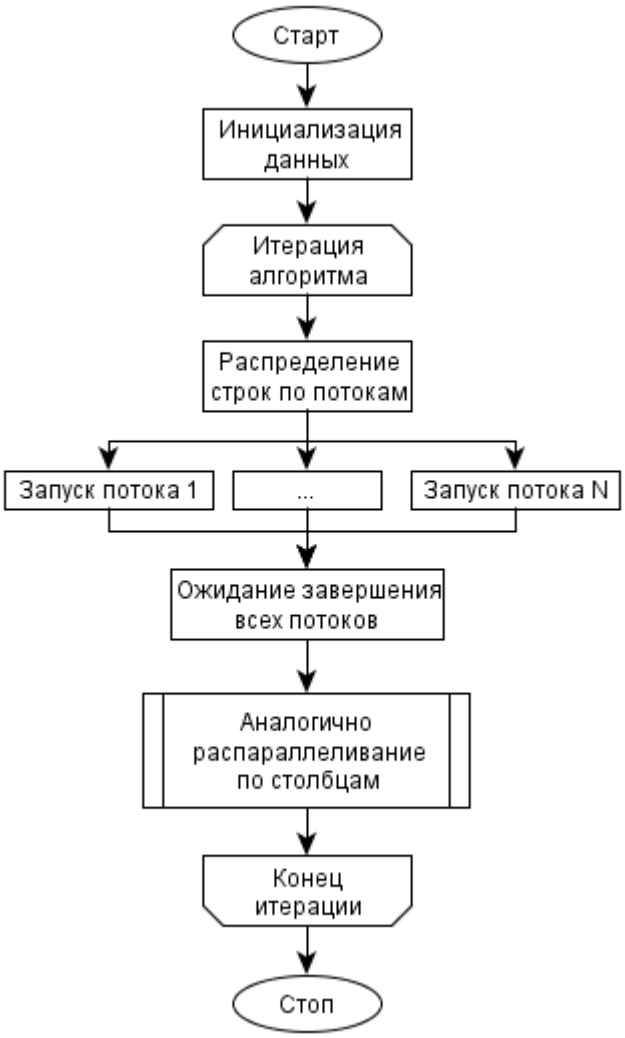

Рис. 1. Схема распараллеливания метода прогонки

На каждой итерации алгоритма сначала создается N потоков для прогонки по строкам (каждый поток вычисляет RowsCount / N строк), основной поток ожидает завершения выполнения каждого из запущенных потоков и запускает N потоков для прогонки по столбцам. После завершения выполнения потоков запускается следующая итерация алгоритма.

Сравнение скорости работы алгоритма с разным числом потоков

Ускорение работы метода при его распараллеливании зависит от нескольких параметров:

1. Количество ядер процессора, на которых может выполняться приложение. При одном ядре потоки будут выполняться последовательно (прерывая друг друга), не

давая выигрыша по производительности. Более того, будет происходить уменьшение производительности из-за накладных расходов на создание потоков. Создавать потоков больше, чем имеется ядер, также не имеет смысла, так как несколько потоков на общем ядре из-за накладных расходов будут выполняться медленнее, чем один поток.

2. Количество ячеек, на которые разбита область решения (пластина). При увеличении размера увеличивается время прогонки по одной строке/столбцу, благодаря чему затраты на накладные расходы (создание потоков) уменьшаются. Таким образом, при увеличении размера пластины должен проявляться выигрыш многопоточной реализации.

При увеличении числа итераций время выполнения как однопоточного, так и многопоточного алгоритма должно линейно увеличиваться, так как потоки создаются на каждой итерации, и время на накладные расходы также линейно увеличивается.

Улучшение производительности при распараллеливании потоков также зависит от выбранного языка программирования, так как разные языки по-разному работают с потоками, и накладные расходы различны.

В данной статье проведены следующие исследования:

1. сравнение времени работы однопоточной и многопоточной реализаций метода в зависимости от числа итераций при постоянном размере пластины;

2. сравнение времени работы в зависимости от размера пластины при постоянном числе итераций.

Исследования были проведены с использованием реализаций на языке C# и Haskell.

На рис. 2 представлено сравнение времени выполнения алгоритма на языке C# в зависимости от числа итераций при постоянном размере пластины (500). Количество потоков 0 обозначает не распараллеленную версию алгоритма. Как видно из рисунка, зависимость от количества итераций линейная. Это значит, что при известном размере пластины достаточно один раз определить, какой из вариантов выполняется быстрее, и использовать его для точных вычислений.

Также было проведено измерение работы алгоритма при создании одного дополнительного потока (на рисунке – красный), который выполнял прогонку по всем строкам. Такой вариант медленнее однопоточной реализации только из-за накладных расходов.

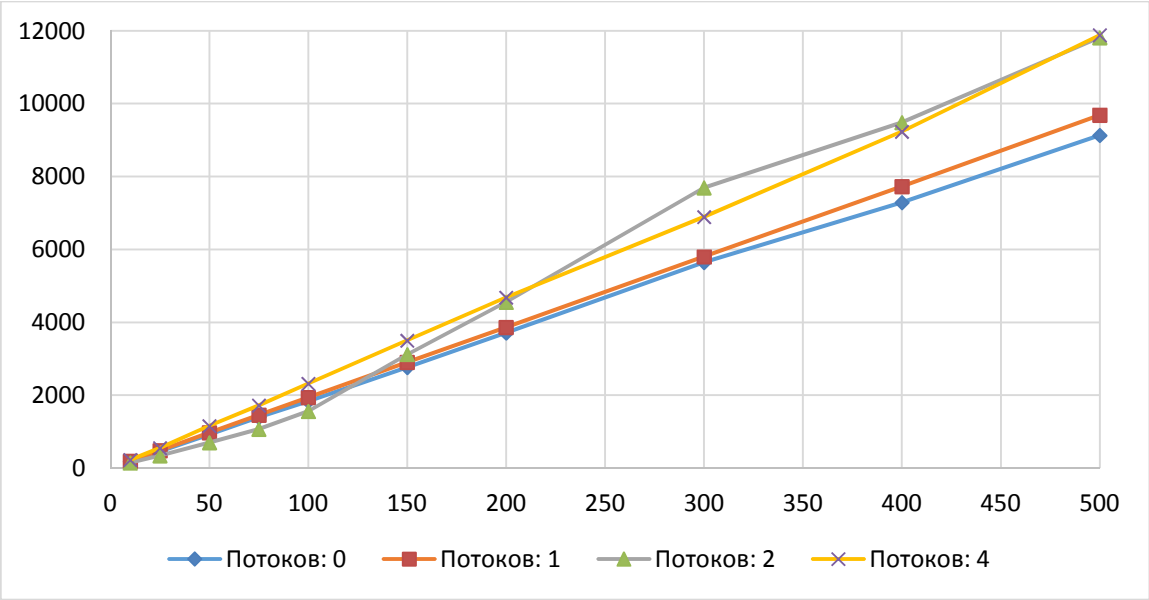

Рис. 2. Зависимость времени выполнения от количества итераций на C#

На рис. 3 представлено сравнение времени выполнения алгоритма на языке Haskell в зависимости от числа итераций при постоянном размере пластины (100). При выполнении 40 итераций происходит резкое возрастание времени выполнения алгоритма. Это связано с реализацией алгоритма, так как при большом числе итераций происходит дополнительное выделение ресурсов программе, что занимает длительное время. Независимо от этого время выполнения однопоточного варианта больше многопоточного при любом числе итераций.

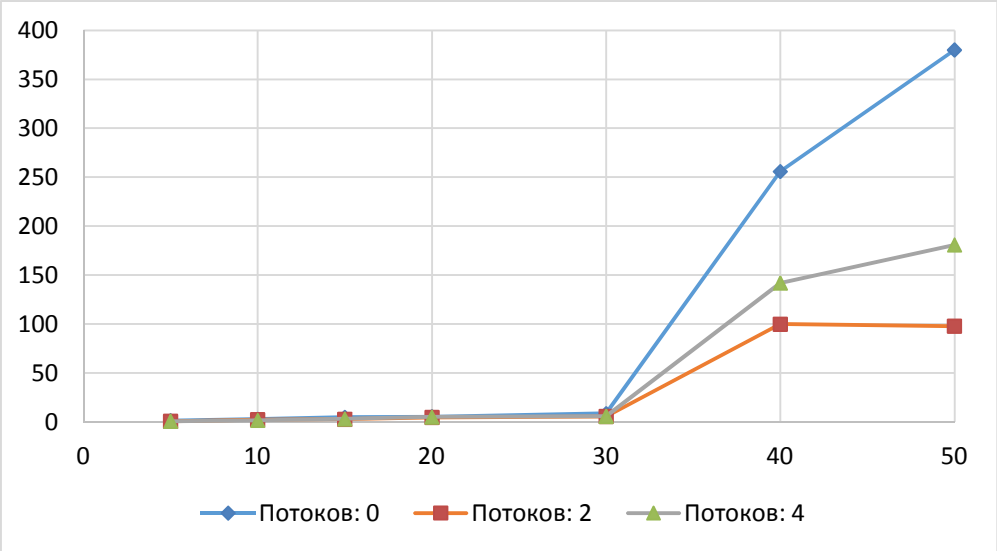

Рис. 3. Зависимость времени выполнения от количества итераций на Haskell

На рис. 4 показана зависимость времени выполнения реализации на языке C# от размера пластины при постоянном числе итераций (200). При маленьком размере

пластины однопоточный метод показывает значительный выигрыш по времени (в 100 раз быстрее при размере 10; в 1.23 раза быстрее при размере 100). При увеличении размера пластины многопоточный вариант начинает работать быстрее (в 1.25 раз быстрее при размере пластины 1500). При этом, несмотря на то, что процессор на тестовом компьютере имеет 4 ядра, двухпоточный вариант лишь немного медленнее четырехпоточного. Это связано как с накладными расходами на создание вдвое большего числа потоков, так и на способ обработки потоков системой (процессорное время не всех ядер может отдаваться потокам приложения). Таким образом, при превышении некоторого размера пластины имеет смысл использовать многопоточный вариант реализации.

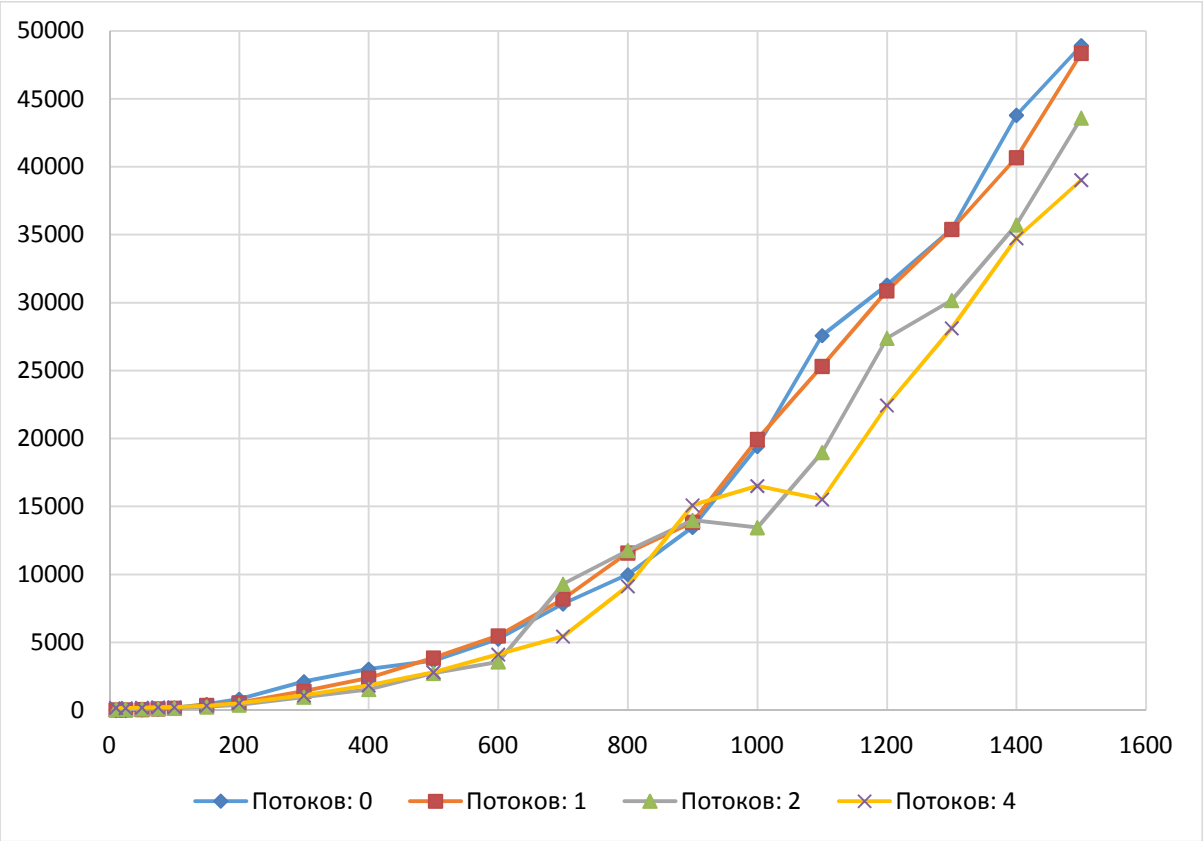

Рис. 4. Зависимость времени выполнения от числа итераций на C#

На рис. 5 показана зависимость времени выполнения реализации на языке Haskell от размера пластины при постоянном числе итераций (15). Как и в реализации на C# многопоточная реализация начинает работать быстрее при увеличении размера пластины. Однако в данном случае рационально использовать многопоточную реализацию уже начиная с размера пластины в 20, тогда как при реализации на C# использовать многопоточный вариант рационально при достижении размером пластины значения в 100.

Это связано с различными способами обработки потоков на разных языках программирования.

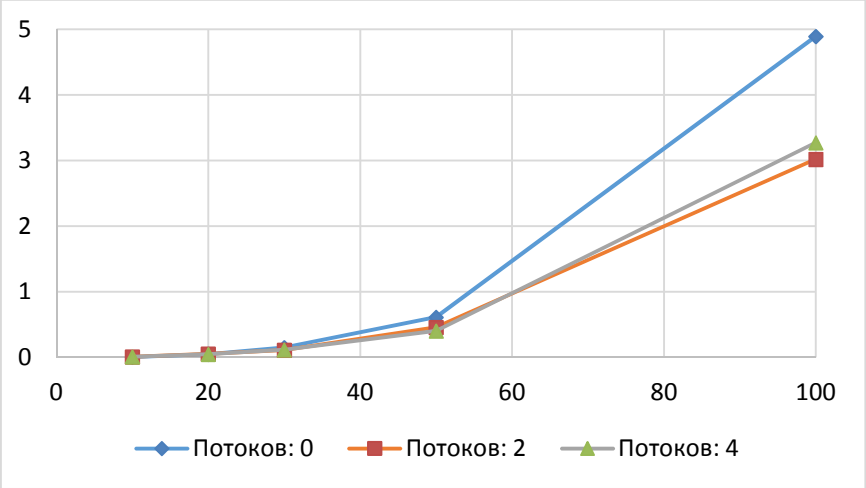

Рис. 5. Зависимость времени выполнения от числа итераций на Haskell.

Заключение

В данной статье был реализован метод продольно-поперечной прогонки с использованием алгоритма распараллеливания вычислений на языках C# и Haskell. Проведены исследования зависимости времени выполнения метода от размера пластины и числа итераций для различного числа потоков. Исследования показали, что многопоточная реализация в C# выгоднее однопоточной при размере пластины больше 100, а в языке Haskell при размере пластины больше 20. При проверке на конкретном компьютере было выявлено, что использования четырех потоков не дает значительного выигрыша по сравнению с использованием двух потоков. Возможно, при увеличении количества ядер процессора большее число потоков будет давать больший прирост производительности.

В реализации на языке Haskell было обнаружено, что при определенном числе итераций происходит резкое возрастание времени выполнения метода. Предположительно, это связано с нехваткой оперативной памяти. Следовательно, увеличение оперативной памяти позволит проводить большее количество итераций без резкого увеличения времени выполнения алгоритма.

Список литературы

1. Калиткин Н.Н. Численные методы. СПб.: БХВ-Петербург, 2011. 592 с.
2. Самарский А.А. Теория разностных схем. М.: Наука, 1989. 656 с.
3. O’Sullivan B., Goerzen J., Stewart D. Real World Haskell. Sebastopol: O’Reilly, 2008. 714 p.