СТАТИЧЕСКИЙ ВРЕМЕННОЙ АНАЛИЗ СИНХРОННЫХ ЦИФРОВЫХ СХЕМ НА МНОГОЯДЕРНЫХ ЭВМ

Н.А. Князев

Введение

В синхронных интегральных схемах оценка быстродействия путем полного моделирования работы схемы слишком трудоемка, и в этом случае используется метод статического временного анализа (СВА) [1]. Во время СВА для каждого узла схемы рассчитываются задержки и другие характеристики сигнала на элементах схемы, выявляются несоответствия между ожидаемым и фактическим временем прихода сигнала [2]. Для СВА синхронная схема представляется в виде направленного графа, в качестве узлов выступают логические элементы, а в качестве дуг межэлементные соединения. На уровне расчета задержки внутри одного элемента используют, как правило, задержку Элмора [3] или асимптотический расчет задержки сигнала [4]. На уровне анализа задержек схемы работает алгоритм обработки взвешенного графа схемы. На сегодняшний день распространен анализ «худшего случая», когда в графе осуществляется поиск путей, порождающих наибольшие несоответствия между ожидаемым и фактическим временем прихода сигнала [1]. В этом случае из всех пришедших на элемент сигналов имеет смысл распространять только сигналы с наибольшими задержками. Чтобы знать характеристики сигналов на всех входах элемента, необходимо рассчитать все элементы, предшествующие данному, что накладывает ограничения на алгоритм обработки графа. В общем случае граф схемы содержит циклы, и тогда задача временного анализа становится NP-сложной, однако существуют методики, приводящие граф схемы к ациклическому [5]. СВА для большой синхронной схемы может занимать более суток, что делает актуальным вопрос об ускорении алгоритмов СВА с использованием параллелизма. В данной работе предлагается многопоточный вариант алгоритма СВА на ациклическом графе. Алгоритм основан на методе «поиска в ширину». В алгоритме учитывается специфика СВА: перед расчетом очередного элемента рассчитываются все предшествующие. Это свойство достигается с помощью механизма счетчиков, отмечающих число входов, для которых не рассчитаны задержки. В случае единичного изменения в схеме предлагается быстрый алгоритм перерасчета задержек. В работе исследуется параллелизуемость и ускорение данных алгоритмов на различном числе ядер.

Постановка Задачи для статического временного анализа

СВА является методом анализа задержки распространения сигнала через схему без использования моделирования работы схемы. Различают анализ «сверху» и анализ «снизу». При анализе «сверху», анализируются максимальные, а при анализе «снизу» минимальные задержки распространения сигнала. В дальнейшем мы будем рассматривать только анализ «сверху», имея в виду, что все рассуждения можно распространить и на анализ «снизу».

Для постановки задачи статического временного анализа введем несколько определений.

1. *Цифровой сигнал* – последовательность импульсов. В зависимости от своих электромагнитных характеристик сигнал может соответствовать различным уровням. Как правило, различают нижний (“0”) и верхний (“1”) уровни.
2. *Событие* – изменение уровня сигнала на входе или выходе элемента. Различают нисходящие (изменение уровня с 1 на 0) и восходящие (изменение уровня с 0 на 1) события.
3. *Задержка сигнала* — время, которое требуется электрическому сигналу, чтобы достичь некоторого узла в схеме, т.е. время до наступления соответствующего данному сигналу события в этом узле схемы.
4. *Запас* — разность между предельно допустимой и фактической задержкой сигнала. Запас может быть отрицательным.
5. *Худший путь* - путь прохождения сигнала от входа до выхода схемы, имеющий наименьший запас.

В статическом временном анализе путь сигнала представляется как упорядоченная последовательность событий в узлах схемы, где одно событие порождает следующее событие в соседнем узле.

Для СВА в качестве входных данных берется:

1. Структура элементов и соединений в схеме.
2. Если ранее производились расчеты данной схемы, то уже рассчитанные пути сигналов.

В общем случае, результат СВА - набор всех путей и их запасов. Число путей может экспоненциально зависеть от размера схемы, поэтому, как правило, рассматривают анализ «худшего случая» (worst-case analysis), где в качестве результата выступает набор худших путей. В ходе этого анализа для каждой точки графа схемы находится путь с самой большой задержкой в нее приходящий. Чтобы не выполнять лишнюю работу – не распространять сигнал с не самой большой задержкой, при расчете элементов используется следующий

алгоритм, называемый прореживанием:

1. Все сигналы со всех входов элемента распространяются на выходы.
2. На каждом выходе выбирается сигнал с самой большой задержкой каждого типа (нисходящий или восходящий).
3. Сигналы с самой большой задержкой с выходов через межсоединения распространяются на входы элементов, соединенных с рассчитываемым элементом.

Более подробно алгоритм описан в [8]

Для анализа задержки элемента необходимо учитывать следующие характеристики сигнала:

1. Задержки сигналов на всех входах элемента.
2. Длительность фронта переключения (время между началом и окончанием изменения сигнала) .
3. Тип событий (нисходящие или восходящие). Некоторые пути в элементе могут быть доступны только для сигналов определенного типа.

Существующие подходы к параллелизации графовых методов статического временного анализа

Как было сказано во введении для методов статического временного анализа на уровне схемы существует ограничение: для корректного анализа элемента необходимо, чтобы сигнал на всех его входах был рассчитан [3]. Современные методы обхода графа схемы для выполнения описанного ограничения проводят анализ вершин в заранее определенном порядке (задают топологический порядок вершин). Наличие топологического порядка гарантирует, что перед анализом элемента значения сигнала на входах элемента будут рассчитаны. Разбиение вершин графа на уровни по удаленности от входных элементов, позволяет рассчитывать элементы на одном уровне параллельно, при помощи метода «поиска в ширину». Принцип разбиения графа на уровни для временного анализа приведен в работах [6] [8] [9]. Параллельная реализация такого подхода описана в работах [6] и [10]. В работе [6] приведен пример параллелизации алгоритма статического временного анализа для многопроцессорных систем. Авторы отмечают, что даже на специально выбранных графах, они сталкиваются с проблемой малого количества элементов на одном уровне. В результате данный подход эффективно применим только для очень «широких» графов (графов, где расстояние от входов до выходов значительно меньше числа вершин). В случае если на одном уровне оказывается малое число вершин, то загрузка процессоров осуществляется неравномерно, ускорение уменьшается.

Пример разделения на уровни показан на рисунке 1. Это разделение не зависит от сложности расчета элемента. Если, к примеру, на вычисление элемента 2 будет тратиться больше времени, чем на вычисление элемента 3, то пока вычисляется элемент 2, остальные вычислительные мощности будут простаивать.

Рис. 1

В статье [7] предлагается параллельный алгоритм СВА без использования строгого прореживания, однако предложенный алгоритм, использует эвристику и не является точным алгоритмом.

Параллельный алгоритм временного анализа на графе без циклов

В параллельных реализациях СВА, в которых используется разделение вершин по уровням, синхронизация различных потоков происходит при переходе с одного уровня на другой. Вычисление нового уровня не начинается, пока не рассчитаны все вершины предыдущего. Такой подход может порождать неравномерную загрузку вычислительных мощностей. В отличие от описанной «синхронной» параллелизации в данной статье предлагается «асинхронная» параллелизация, когда вместо разделения вершин по уровням, элемент начинает рассчитываться, как только для него готовы данные. Для обеспечения асинхронной параллелизации каждому элементу ставится в соответствие счетчик, соответствующий числу входов, на которых еще не готовы события сигнала. Изначально счетчики всех элементов кроме начальных устанавливаются равными количеству входов. В процессе временного анализа этот счетчик уменьшается. Когда счетчик равен 0, элемент готов к расчету. Данный подход дает возможность параллельно рассчитывать элементы со счетчиком, равным нулю. Алгоритм в таком случае не ждет, пока будут рассчитаны все элементы соответствующего уровня, а сразу приступает к расчету элементов, данные для которых готовы. Предлагаемый алгоритм состоит из двух этапов, каждый из которых можно реализовать с использованием многопоточного

программирования. Основой каждого из этапов является обход графа с помощью «поиска в ширину».

Первый этап. Инициализация счетчиков.

На этом этапе необходимо на всех элементах схемы инициализировать счетчики числом входов. Для временного анализа достаточно установить счетчики только на элементах, достижимых из начальных. В данном случае может использоваться поиск в ширину из начальных элементов. Значение счетчика для всех элементов по умолчанию равно 0. Выполняя «поиск в ширину», значение счетчика увеличивается каждый раз, когда мы обнаруживаем путь из начальных элементов до данного. Если при обнаружении пути значение счетчика элемента было равно 0, то этот элемент попадает в очередь и осуществляется поиск путей, проходящих через этот элемент. В общем случае есть возможность, что одни начальные элементы достижимы из других, однако проверять каждый раз, является ли элемент начальным, затратно. В качестве оптимизации до начала обхода графа схемы счетчик всех входных элементов увеличивается на 1. Таким образом, они не могут повторно попасть в очередь. После стадии расстановки счетчиков для входных элементов счетчик обратно уменьшается на 1. При этом для всех начальных элементов, у которых счетчик после выполнения алгоритма остается больше 1, известно, что они достижимы из других начальных элементов на графе схемы. Такие элементы нужно удалить из начального списка элементов.

Таким образом, здесь выполняется простой обход элементов, который может быть реализован следующим образом:

1. Для всех элементов счетчик равен 0.
2. Увеличить у всех начальных элементов счетчик на 1.
3. Добавить начальные элементы в очередь.
4. Извлечь какой либо элемент из очереди.
5. Для каждого элемента, вход которого соединен с выходом данного, выполнить:
   a) Если счетчик равен 0, то добавить элемент в очередь.
   b) Увеличить счетчик на 1.
6) Если очередь пуста, перейти к пункту 6 иначе к пункту 3.
7) Для всех начальных уменьшить счетчик на 1.
8) Удалить все начальные элементы с не нулевым счетчиком.

Количество увеличений счетчика на этом этапе алгоритма равно количеству узлов в схеме. Однако время работы этого алгоритма существенно меньше времени работы этапа расчета элементов.

Второй этап. Расчет элементов.

На этом этапе производится непосредственно расчет схемы, т.е. распространение событий от входов схемы к ее выходам и расчет худших путей. Физический расчет элементов выполняется при помощи библиотеки RICE [11] и остается за пределами данной статьи.

После выполнения этапа расстановки счетчиков алгоритм представляет собой следующую последовательность действий:

1. Поместить в очередь начальные элементы.
2. Выбрать элемент из очереди.
3. Рассчитать элемент, распространить события со входов элемента на его выходы.
4. Осуществить прореживание сигналов.
5. Рассчитать межсоединения, распространить события с выходов элемента на входы элементов, соединенных с выходами данного.
6. Для каждого элемента, входы которого соединены с выходами данного, уменьшить счетчик на 1.
7. Если счетчик какого-либо из элементов стал равен 0, то добавить его в очередь.
8. Если очередь пуста, закончить алгоритм, иначе перейти на пункт 2.

В данном алгоритме рассчитывать и добавлять элементы в очередь можно в параллельном режиме.

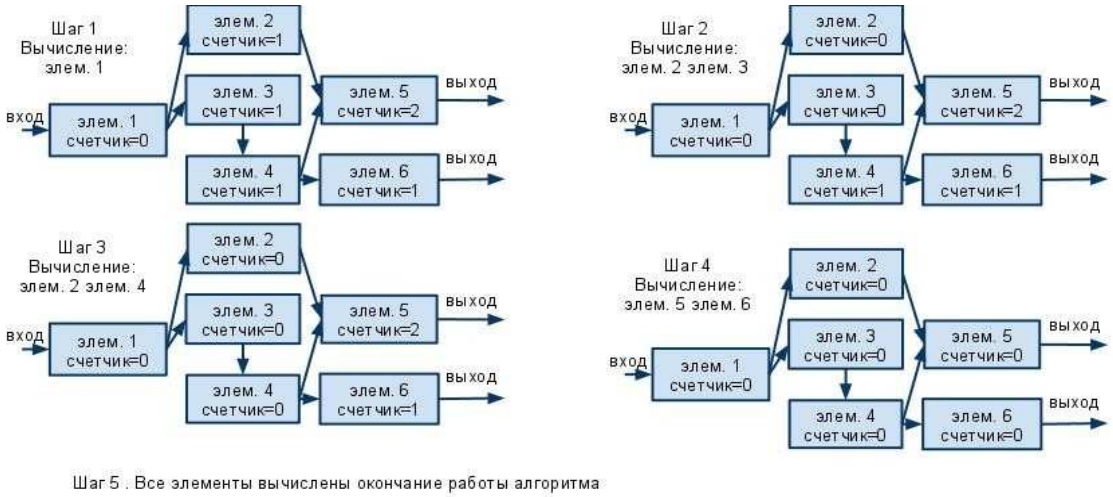

Рис. 2

Пример функционирования алгоритма на двухъядерном компьютере после расстановки счетчиков на простой схеме рисунке 1., показан на рисунке 2. Данный подход решает проблему, когда на вычисление элемента 2 тратится больше времени, чем на другие элементы: в этом случае второе ядро не простаивает, а сразу приступает к вычислению элемента 4.

Пример распределения заданий по процессорным ядрам, при синхронном и асинхронном подходе на простом примере схемы (см. рисунок 1. и рисунок 2), показано на рисунке 3.

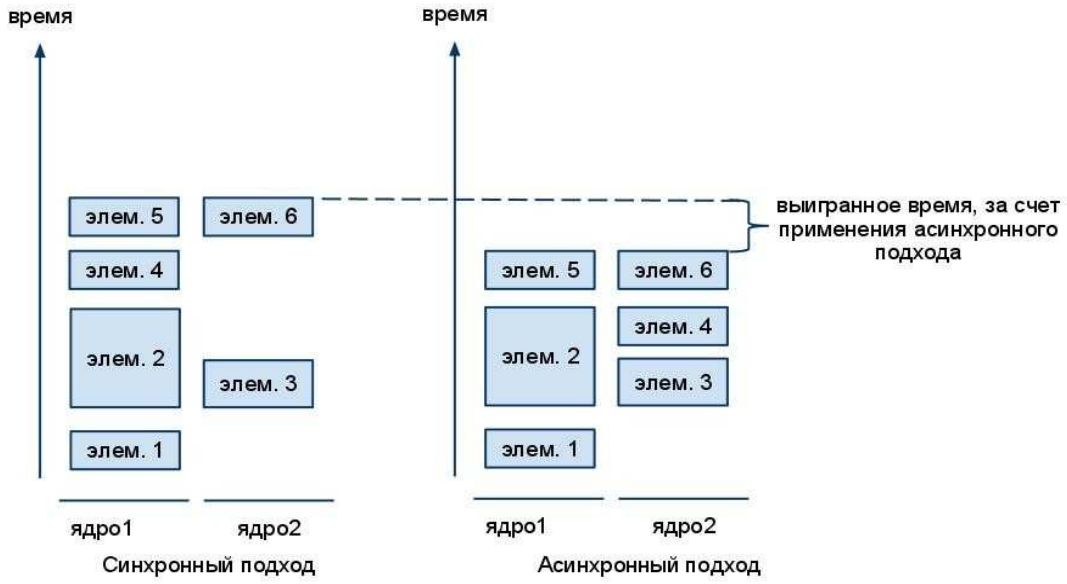

Рис. 3

Оценка сложности предлагаемого алгоритма

Сложность алгоритма складывается из сложностей его составных частей.

Пусть количество достижимых элементов в схеме N. Как было сказано ранее, инициализация счетчиков занимает намного меньше времени, чем расчет схемы. Во время второго этапа алгоритма каждый элемент вычисляется только один раз, таким образом, сложность равна O(N).

Инкрементальный параллельный алгоритм временного анализа после единичного изменения.

После изменения одного элемента в схеме, во временном анализе возникает задача быстрого перерасчета событий на элементах. На практике часто допускают, что структура худших путей не изменилась. В этом случае выполнять выше описанный алгоритм нет необходимости, можно ограничиться перерасчетом уже известных путей сигнала. Для отражения этого изменения достаточно перерассчитать все пути проходящие через измененный элемент. Таким образом, изменение одного элемента порождает изменение событий в конусе схемы с вершиной в измененном элементе (см. рисунок 4).

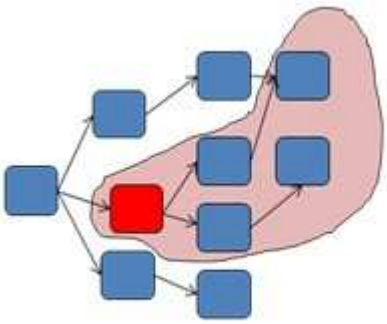

Рис. 4

Перерасчет событий на конусе схемы намного более простая задача, чем полный статический временной анализ. Инкрементальный перерасчет конуса можно делать в интерактивном режиме, в то время как полный анализ занимает часы. В этом случае можно использовать простой алгоритм «поиска в ширину», по всем временным путям. Алгоритм будет выглядеть следующим образом:

1. Поместить в очередь все рассчитанные события на измененном элементе.
2. Извлечь событие из очереди.
3. Перерассчитать событие.
4. Добавить в очередь все события, порождаемые перерассчитанным.
5. Если очередь пуста, то закончить алгоритм, иначе перейти на пункт 2.

В данном алгоритме также используется поиск в ширину, однако, в отличие от описанного алгоритма СВА на графе без циклов, нет этапа расстановки счетчиков. Этот этап здесь не нужен, т.к. граф путей представляет собой дерево (у каждого события только одно порождающее) и синхронизации для выполнения прореживания не требуется. В данном алгоритме обрабатывать и добавлять элементы в очередь можно в многопоточном режиме, что делает возможным реализацию алгоритма для многоядерных ЭВМ.

Реализация алгоритмов

Предложенный алгоритм реализован на языке C++ с использованием библиотеки для многопоточного программирования Intel Threading Building Blocks (TBB) [12]. Алгоритмы асинхронного поиска в ширину и перерасчета событий реализованы в многопоточном режиме с помощью шаблона TBB parallel_do. Синхронизация для алгоритма СВА на ациклическом графе происходит в момент уменьшения счетчика, таким образом, какой-либо синхронизации в момент моделирования элемента не требуется. Изменение счетчиков происходит при помощи механизма аппаратно неделимых атомарных операций с использованием шаблона TBB atomic. Для инкрементального алгоритма перерасчета событий на конусе схемы синхронизация не требуется вовсе. Для реализации кэшей и инкрементальной загрузки данных схемы с диска в условиях разделяемой памяти многоядерной системы использовались шаблоны TBB concurrent_hash_map, concurrent_vector, mutex spin_lock и другие.

Результаты

Предложенный алгоритм был протестирован на 16ти ядрах Intel Xeon. Ввиду того, что загрузка данных о схеме выполняется последовательно, ускорение зависит от того загружены ли данные в оперативную память (данные могут быть уже загружены в оперативную память, например, при повторном запуске алгоритма). Максимальное ускорение, которое удалось достичь на 16 ядрах – 13 раз. Максимальное ускорение реализации синхронного алгоритма на многопоточной системе составило 11раз на 16 потоках. Это позволяет сделать вывод о эффективности предложенного метода асинхронного поиска в ширину.

Тестирование показало, что предложенная в данной статье реализация алгоритма позволяет добиться большего ускорения, чем реализация, предложенная в статье [7]. Согласно авторам статьи [7] максимально достигнутое ускорение их алгоритма 4.52 раза, ускорение уменьшается с ростом числа потоков. Параллельный алгоритм СВА на ациклическом графе, представленный в работе [10] реализован для многопроцессорной системы IBM BlueGene/L, на которой достигнуто 10-кратное ускорение на 16 потоках.

Ускорение для предложенного алгоритма расчета задержки на ациклическом графе изображено на рисунке 5.

Рис. 5

Ускорение для алгоритма пересчета схемы после единичного изменения показано на рисунке 6. В этом случае ускорение также зависит от предварительной загрузки данных для схемы. Максимальное ускорение, которое удалось достичь 16 ядрах – 14.5 раз.

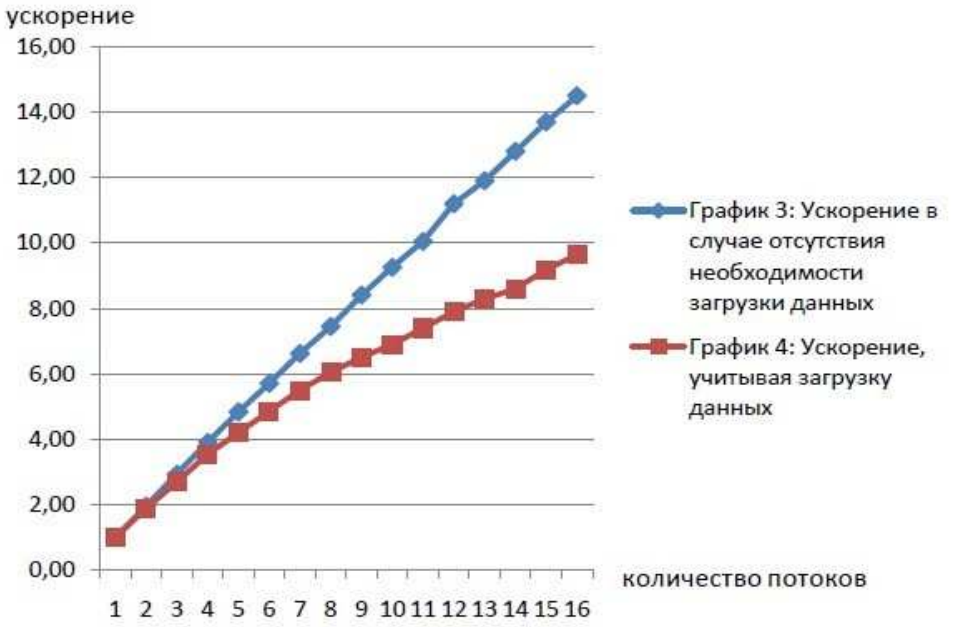

Рис. 6

Параллелизация загрузки данных схемы может приблизить график 2 к графику 1 и график 4 к графику 3.

ЛИТЕРАТУРА

1. Гаврилов С., Стемпковский А., Глебов А. "Методы логического и логико-временного анализа цифровых СБИС"// Москва, 2007, изд. Наука.
2. Soha Hassoun, Tsutomu Sasao. "Logic synthesis and verification". //Springer 2002. ISBN 978-0-7923-7606-4.
3. Rakesh Chadha, J. Bhasker, "Static Timing Analysis for Nanometer Designs, A Practical Approach." // Springer, 2009. ISBN 978-0-387-93819-6.

4. Lawrence T.Pillage and Ronald Rohrer "Asymptotic Waveform Evaluation for Timing Analysis"// 1990, Transaction on computer-aided design, IEEE.
5. Jin-fuw Lee, Donald T. Tang "An Algorithm for Incremental Timing Analysis"// 1995, 32nd ACM/IEEE Design Automation Conference
6. Akintayo Holder, Christopher D. Carothers, Kerim Kalafala. "Prototype for a Large-Scale Static Timing Analyzer running on an IBM Blue Gene." // 2010, IEEE ISBN 78-1-4244-6534-7
7. Li-Ren Liu, David H.C. Du Hsi-Chuan Chen, "An Efficient Parallel Critical Path Algorithm" // 1991,Department of Computer Science University of Minnesota Minneapolis
8. Валентин Ирбенек, "Временная верификация и оптимизация размещения компонентов предельных по быстродействию ЭВМ"// Москва, 2001, дис. доктора технических наук, Институт микропроцессорных вычислительных систем Российской академии наук.
9. Blaauw David, Zolotov Vladimir, Sundareswaran Savithri, "Slope Propagation in Static Timing Analysis" // 2002, Transactions on Computer-Aided Design of Integrated Circuits & Systems;
10. Richard E. Korf and Peter Schultze. "Large-Scale Parallel Breadth-First Search"// 1994, University of California, Los Angeles.
11. C. Ratzlaff and L. Pillage , "RICE rapid interconnect circuit evaluation" // 1994, Computer-Aided Design of Integrated Circuits and Systems,
12. Threading Building Blocks Reference Manual// 2010, [сайт] URL:http://www.intel.com. 2010.