УДК 621.38:519.178

Параллельный алгоритм поиска критических путей и циклов в задаче статического временного анализа цифровых схем с последовательностной логикой

Н.А. Князев[1,2], К.К. Малинаускас[1]

[1]ЗАО «Интел А/О», nickolai.a.knyazev@intel.com, kostas.malinauskas@intel.com

[2]Институт Прикладной Математики им. М.В. Келдыша РАН

***Аннотация* — Поиск критических путей является одним из ключевых этапов статического временного анализа и временной верификации цифровых схем. Для содержащих несколько миллионов транзисторов СБИС время работы алгоритмов временного анализа может достигать нескольких суток, что существенно осложняет разработку схем. В данной работе предлагается параллельный алгоритм, позволяющий с использованием многоядерных ЭВМ на порядок ускорить поиск критических путей в СБИС. При этом поддерживается анализ схем последовательностной логики и обнаружение критических циклов, что расширяет предложенный ранее авторами параллельный алгоритм для комбинационных схем. Используемая асинхронная параллелизация позволяет сбалансировать нагрузку на вычислительные ресурсы. Нам удалось достичь ускорения до 8.9 раз на 16 ядрах относительно последовательной реализации.**

***Ключевые слова* — параллельные алгоритмы, поиск критических путей в графе, статический временной анализ.**

I. Введение

Статический временной анализ (СВА) – распространенный подход для оценки быстродействия синхронных цифровых схем. Полное моделирование переходных процессов в СБИС слишком трудоемко, поэтому СВА получил широкое распространение на практике. В статическом временном анализе задержки сигналов вычисляются путем упрощенного моделирования отдельных элементов схемы и использования сценариев худшего случая распространения сигналов. Основной задачей СВА является выявление критических временных путей и определение максимально допустимой тактовой частоты работы схемы [1]. В данной работе предлагается параллельный алгоритм поиска критических путей для схем с последовательностной логикой. За рамками рассмотрения остаются такие проблемы, как выявление «ложных путей» [1]-[3] и поиска околокритических путей [2], [4].

В дальнейшем будем использовать следующие определения. *Событие* – изменение уровня сигнала (логического «0» или «1») в узле схемы (на входе или выходе логического элемента). *Временной граф* цифровой схемы – направленный граф, где в качестве вершин выступают узлы схемы, а направленные дуги соответствуют причинно-следственным связям между возможными событиями в соседних узлах. *Задержка события* – время наступления события относительно некоторого начала отсчета. События отсчитываются от моментов переключения идеального генератора тактовых импульсов. *(Временной) путь* – последовательность событий, где очередное событие порождается предыдущим в смежном узле. *Задержкой пути*, заканчивающегося событием, называется задержка этого события. *Критический путь* – это путь, заканчивающийся событием с наихудшей задержкой среди всех однотипных событий в данном узле схемы. *Однотипными* называются события, у которых совпадают начала отсчета и направления изменения уровня сигнала. *Замкнутый путь* – это путь, заканчивающийся в уже пройденном ранее узле. *Критический цикл* – это замкнутый критический путь.

Во время СВА осуществляется распространение событий от входов к выходам схемы. Число всех возможных путей может экспоненциально зависеть от размера схемы, поэтому часто рассматривают анализ *худшего случая* [2], [5], [6], где в качестве результата выступает набор критических путей. При этом различают анализ задержек «сверху» и «снизу», соответственно рассматривая в качестве критических путей самые медленные и самые быстрые временны*е* пути. Для этого при распространении событий используется *прореживание*: в каждом узле схемы среди однотипных событий выбираются события с наихудшей задержкой (максимальной при анализе «сверху», минимальной при анализе «снизу»), и далее по схеме распространяются только они [7].

Разными по сложности являются задачи СВА для комбинационных схем и для схем, содержащих последовательностную логику. Задача для комбинационных схем сводится к поиску критических путей в направленном ациклическом графе и решается за линейное время распространением событий в топологическом порядке [2]. Наличие триггеров может привести к образованию циклов в схеме (см. рис. 1). Если временной граф содержит циклы, то использование топологического порядка становится

невозможно и применяются более сложные алгоритмы. В присутствии критических циклов задача поиска критических путей становится неопределенной, так как всегда найдется путь со сколь угодно худшей задержкой. Поэтому обычно при обнаружении критических циклов алгоритмы СВА сообщают о них пользователю и продолжают анализ с разрывом циклов (например, путем игнорирования некоторых дуг графа).

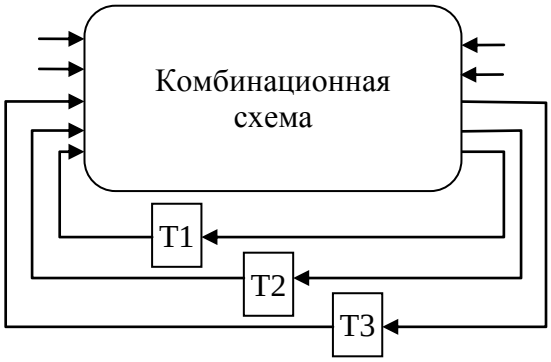

Рис. 1. Цикличная схема с триггерами

В правильно спроектированных схемах к циклам во временном графе могут приводить только триггеры со статическим управлением (тактируемые уровнем). В них возможны два варианта распространения событий. В режиме «прозрачности» (между открывающим и закрывающим событиями на входе синхронизации) события со входа данных распространяются на выход триггера. В режиме «непрозрачности» триггер закрыт для передачи данных на выход до прихода очередного открывающего события. То есть событие на выходе будет порождено событием на входе синхронизации. В триггерах с динамическим управлением (тактируемых фронтом) возможен только второй вариант распространения событий. Различным вариантам соответствуют дуги во временном графе, как показано на рис. 2.

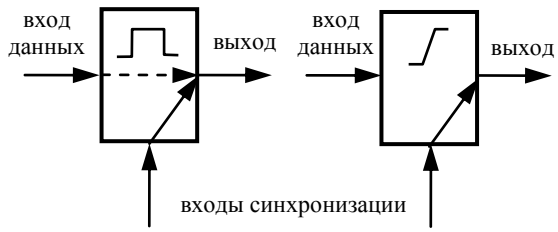

Рис. 2. Временной граф для триггеров со статическим (слева) и динамическим (справа) управлением

При распространении через триггер задержка события переопределяется, изменяя точку отсчета на точку отсчета закрывающего триггер события синхронизации. Для этого используется оператор фазового сдвига [8]. Поскольку временной путь может проходить через несколько «прозрачных» триггеров, то он может оказаться замкнутым. После применения операторов фазового сдвига замкнутый путь не обязательно является критическим. Если же замкнутый путь критический, то в процессе дальнейшего его распространения по циклу задержка накапливается, что, в конечном счете, приводит к нарушению корректной работы схемы. Поэтому обнаруженные критические циклы необходимо устранять при проектировании схемы.

В работах [8], [9] рассматриваются алгоритмы СВА для графов с циклами. Для сверхбольших интегральных схем временной анализ может занимать более суток [10], [11], что существенно осложняет их разработку. Частично проблему быстродействия решают инкрементальные алгоритмы [12], позволяющие более быстро переанализировать схему в случае небольших ее локальных изменений.

В общем случае повысить быстродействие СВА можно с помощью параллельных вычислений. Известны реализации СВА для распределенных вычислительных систем. Например, в [11] предлагается подход с использованием ранжирования вершин графа по максимальной удаленности от входных вершин. Элементы одного уровня анализируются параллельно с синхронизацией при переходе на последующий. Однако метод использует топологический порядок, и применим лишь к ациклическим графам.

В настоящее время активно развиваются многопроцессорные и многоядерные системы с общей памятью. Однако в области СВА ощущается недостаток работ на тему параллелизации для таких систем. Ранее нами был предложен параллельный алгоритм поиска критических путей в ациклических временных графах, реализованный с использованием многопоточности [10]. Алгоритм также использует топологический порядок и не применим к схемам, содержащим триггеры со статическим управлением. Мы показали, что без использования ранжирования достигается лучшая балансировка загрузки вычислителей.

В данной работе предлагается параллельный итерационный алгоритм поиска критических путей и циклов для схем с последовательностной логикой, расширяющий предложенный нами ранее алгоритм для комбинационных схем [10]. Нам удалось достичь ускорения многопоточной реализации алгоритма до 8.9 раз на 16 ядрах.

II. Используемые технологии параллельного программирования

В качестве основного инструмента мы использовали библиотеку Intel ® Threading Building Blocks (TBB) для языка C++ [13], предоставляющую высокоуровневые шаблоны и примитивы параллельного программирования для многопроцессорных систем с общей памятью. Основной концепцией TBB является абстрагирование от исполняемых потоков и архитектуры конкретной вычислительной системы. Для достижения высокой эффективности параллелизма и масштабируемости программы от разработчика требуется разбить вычислительную проблему на множество мелких задач. Обычно это легко достигается при реализации

параллелизма по данным. Встроенный в TBB динамический планировщик задач назначает их на исполняемые потоки, используя технику «воровства задач» для балансировки загрузки (простаивающие потоки забирают задачи у перегруженных). Кроме того, библиотека TBB оптимизирована под использование кэш-памяти.

В реализации наших алгоритмов мы использовали шаблон TBB parallel_do, который позволяет добавлять задачи в общий пул, в том числе в процессе выполнения задач. В качестве неделимой задачи выступает временной анализ отдельного логического элемента и распространение событий на смежные с ними элементы. Также мы использовали такие примитивы TBB, как потокобезопасные контейнеры, мьютексы (примитивы для предотвращения одновременного доступа к общим ресурсам), атомарные операции, масштабируемые распределители памяти.

III. Параллельный алгоритм СВА для комбинационных схем

В параллельных реализациях СВА, использующих ранжирование по уровням (степеням максимальной удаленности от входов схемы), анализ нового уровня не начинается, пока не готовы события на предыдущем [11]. Ранее мы показали, что такой подход может порождать неравномерную загрузку вычислителей, и предложили «асинхронный» алгоритм [10], обладающей большей эффективностью параллелизации.

В «асинхронном» подходе логический элемент начинает анализироваться, как только для него готовы входные события. Каждому элементу ставится в соответствие счетчик, соответствующий числу входных связей, по которым еще не проанализированы худшие задержки событий. В процессе временного анализа этот счетчик уменьшается. Когда его значение достигает нуля, элемент готов к анализу и становится активным. Данный подход дает возможность параллельно проводить временной анализ активных элементов. Параллельная реализация осуществлена с помощью шаблона parallel_do библиотеки TBB. Параллельно производится как временной анализ элементов, так и предварительная инициализация счетчиков.

Под начальными элементами понимаются логические элементы, соединенные с источниками цифровых сигналов, события на которых необходимо распространить по схеме. В процессе выполнения определяются лишние начальные элементы – достижимые из других начальных элементов. Отметим, что алгоритм является инкрементальным для случая локального ухудшения задержек на отдельных элементах. Если критические пути в схеме уже найдены, и на отдельных элементах задержки ухудшились, то можно эти элементы задать в качестве начальных. Тогда алгоритм обновит только изменившиеся критические пути в схеме. Данный факт используется позже в предлагаемом нами итерационном алгоритме СВА для схем с последовательностной логикой.

Алгоритм СВА для комбинационных схем состоит из двух этапов – инициализация счетчиков и временной анализ логических элементов. Рассмотрим их подробнее.

A. *Инициализация счетчиков*

Вход: граф схемы, список начальных элементов; счетчики всех элементов равны нулю.

Выход: счетчики всех элементов равны количеству входящих связей из элементов, достижимых из начальных; из списка начальных элементов удалены лишние элементы.

1. Увеличить на 1 счетчик для всех начальных элементов.
2. Добавить начальные элементы в пул.
3. Пока пул не пуст, параллельно выполнять:
   a. Извлечь произвольный элемент из пула.
   b. Для каждого элемента, соединенного с выходом данного:
      i. Если счетчик равен 0, добавить элемент в пул.
      ii. Увеличить счетчик на 1.
4. Уменьшить на 1 счетчик для всех начальных элементов.
5. Удалить все начальные элементы с ненулевым значением счетчика.

Данный этап является подготовительным для проведения временного анализа в топологическом порядке. Для каждого логического элемента вычисляется начальное значение счетчика, равное числу входящих связей из элементов, достижимых из начальных. Шаги 1, 6 и 7 предназначены для удаления лишних начальных элементов из списка. Шаг 3 выполняется с помощью параллельного шаблона TBB parallel_do. Изменения значений счетчика реализуется как атомарные операции.

B. *Анализ элементов*

Вход: граф схемы, список начальных элементов (без лишних), подготовленные счетчики элементов.

Выход: набор критических путей.

1. Поместить в пул начальные элементы.
2. Пока пул не пуст, параллельно выполнять:
   a. Извлечь произвольный элемент из пула.
   b. Провести временной анализ логического элемента и выходных соединений.
   c. Распространить события со входов элемента на выходы, выполняя прореживание.
   d. Распространить события с выходов элемента на входы смежных элементов, выполняя прореживание.
   e. Уменьшить на 1 счетчик каждого смежного элемента.
   f. Добавить в пул смежные элементы, счетчики которых равны 0.

На данном этапе алгоритма производится временной анализ схемы в топологическом порядке, т.е. распространение событий от входов схемы к ее выходам и вычисление критических путей. Использование счетчиков, проинициализированных на предыдущем этапе, гарантирует, что к моменту начала анализа элемента схемы худшие задержки на его входах уже найдены. Шаг 2 выполняется с помощью параллельного шаблона TBB parallel_do. Изменения значений счетчика реализуется как атомарные операции.

IV. Параллельный итерационный алгоритм СВА для схем с последовательностной логикой

В данной главе предлагается параллельный алгоритм для статического временного анализа схем с последовательностной логикой. Он представляет собой итеративное выполнение параллельного алгоритма для комбинационных схем и распространение событий через триггеры со статическим управлением, которое также осуществляется параллельно.

A. Общий вид алгоритма

В алгоритме используется свойство схем, заключающееся в том, что любой цикл содержит хотя бы один последовательностный элемент (триггер). Таким образом, запретив распространение событий на всех триггерах, временной граф схемы приводится к ациклическому. На самом деле достаточно запретить распространение событий со входов данных на выход триггеров со статическим управлением. Однако здесь и далее, для простоты изложения будем считать, что ациклический временной граф строится путем полного исключения всех триггеров.

В общем виде блок-схема алгоритма изображена на рис. 3. Статический временной анализ является итерационным процессом с чередующимися этапами анализа ациклической части графа без триггеров и анализа только триггеров. Для вычисления критических путей в ациклическом графе используется описанный выше алгоритм для комбинационных схем. После его выполнения анализируются триггеры, на входах которых изменились худшие задержки. На этом этапе события распространяются через триггеры. Элементы ациклического графа, на входах которых задержки изменились (ухудшились), используются в качестве начальных для повторного выполнения алгоритма для комбинационной логики. Итерации повторяются, пока худшие задержки не стабилизируются. Обнаружение и разрыв критических циклов будут рассмотрены ниже.

Отметим, что итерационный алгоритм является инкрементальным для случая локального ухудшения задержек на отдельных элементах. То есть, если худшие пути уже вычислены, а затем на некоторых элементах задержки ухудшились, то эти элементы можно задать в качестве начальных. Алгоритм тогда обновит только те критические пути, на которые повлияло локальное ухудшение задержек. Случай локального улучшения задержек требует небольшой модификации алгоритма и в данной работе не рассматривается.

Как и временной анализ ациклического графа, этап распространения событий на триггерах мы производим параллельно, с использованием шаблона TBB parallel_do.

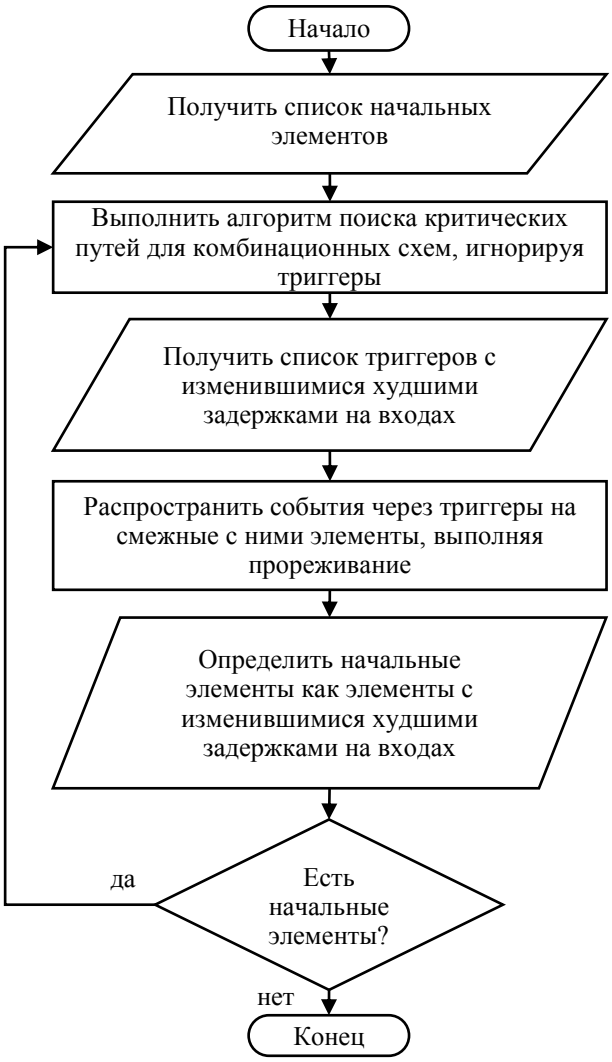

Рис. 3. Блок-схема алгоритма поиска критических путей для схем с последовательностной логикой

B. Обнаружение циклов

Обнаружение циклов происходит на этапе прореживания событий. Когда в узле появляется событие с задержкой худшей, чем у всех имеющихся однотипных событий, осуществляется проверка на цикл. Для этого просматривается назад путь, оканчивающийся новым событием. Если он посещает текущий узел повторно, то это означает, что обнаружен критический цикл. В противном случае новый путь становится критическим.

Для разрыва цикла мы помечаем первый встреченный триггер при обратном просмотре как

запрещенный. В дальнейшем события на запрещенных триггерах не распространяются.

C. Пример работы

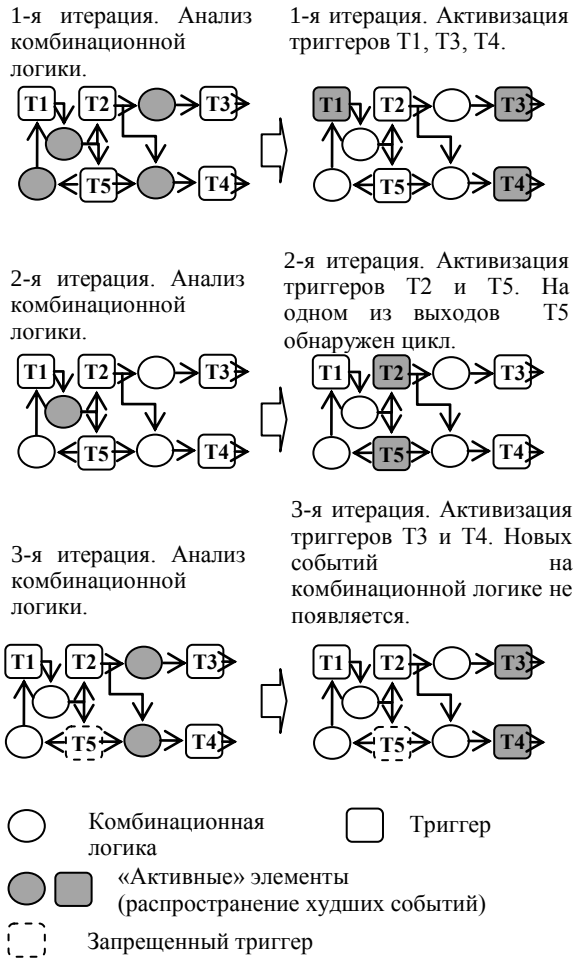

Рис. 4. Пример работы итерационного алгоритма

Пример работы алгоритма для простейшей схемы с пятью триггерами и одним циклом проиллюстрирован на рис. 4. Для упрощения распространение событий в синхронизирующих цепях опущено, начальные события задаются сразу на выходах всех триггеров. Этапы анализа комбинационной и последовательной логики чередуются. Элементы «активизируются», когда на их входы поступают новые значения худших задержек событий. Триггер «активизируется», только если событие на входе попадает во временное окно «прозрачности» триггера. Критический цикл обнаруживается на третьей итерации при анализе комбинационной схемы. Далее по циклу события не распространяются, а один из триггеров, входящий в цикл, помечается как запрещенный для дальнейшего анализа. Алгоритм завершает работу при стабилизации значений худших задержек в схеме. Сходимость достигается за три итерации, что соответствует наибольшему числу триггеров, входящих в критический путь. В данном примере самый длинный путь из триггеров – T1-T5-T4.

V. Анализ и результаты

Наиболее трудоемкий этап алгоритма поиска критических путей на комбинационных схемах – временной анализ логических элементов. Каждый элемент, достижимый из начальных, в данном алгоритме анализируется один и только один раз. Таким образом, сложность алгоритма – $O(N)$, где N – количество элементов в схеме. Максимальное достигнутое нами ускорение – 14 раз на 16 вычислительных ядрах. Более подробный анализ представлен в [10].

Сложность одной итерации алгоритма для схем с последовательностной логикой складывается из сложности алгоритмов для комбинационной логики и временного анализа активных триггеров. Таким образом, сложность итерации оценивается как $O(N+M)$, где N – количество элементов комбинационной логики, а M – количество триггеров. Количество итераций алгоритма равно числу триггеров L в самом длинном критическом пути схемы. Следовательно, суммарная сложность алгоритма – $O((N+M)*L)$.

Экспериментальные результаты по ускорению параллельного алгоритма на многоядерном компьютере для тестовых схем представлены на рис. 5 и в табл. 1.

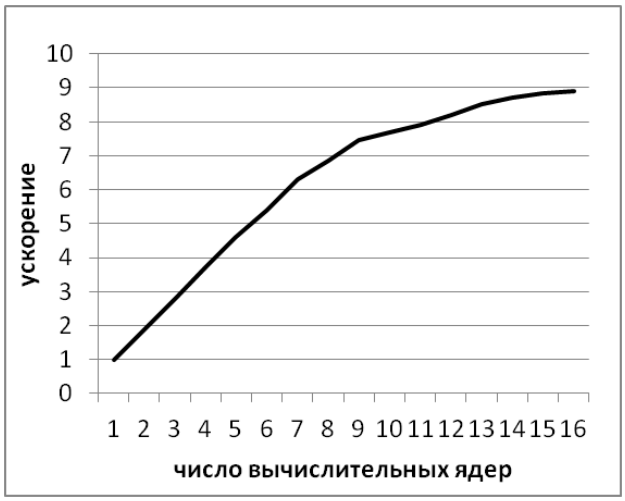

Рис. 5. Ускорение параллельного итерационного алгоритма (на схеме из $N+M$=6756 элементов, включая M=138 триггеров, L=3 триггеров в самом длинном пути).

Эффективность параллелизма существенно зависит от вида и размера схем. Использование топологического порядка ограничивает возможности параллелизма. Поэтому ускорение выше для «широких» схем, содержащих намного больше узлов, чем самый длинный временной путь. В таких схемах большее число элементов может анализироваться независимо и параллельно. На практике, как правило, параллелизм СВА тем эффективнее, чем больше размер схемы. В целом это подтверждается серией экспериментов и результатами, приведенными в табл. 1.

Таблица 1

Экспериментальные данные

Общее число элементов $N+M$	Число триггеров M	Число итераций алгоритма L	Время работы на 1 ядре (сек)	Время работы на 16 ядрах (сек)	Ускорение
512	108	3	8.9	3.3	2.7
1772	177	4	42.0	8.3	5.0
3244	350	4	65.0	18.4	3.5
3573	319	4	123.0	16.9	7.3
5228	390	3	144.0	26.0	5.5
5465	691	4	143.0	28.0	5.8
6756	138	3	203.0	23.0	8.9
19480	1528	3	333.4	43.0	7.8
107000	9160	4	1299.0	175.6	7.4

VI. Заключение

Проблема производительности средств статического временного анализа актуальна для современных СБИС. Авторами разработан параллельный алгоритм поиска критических путей в задаче СВА. Особенностью алгоритма является эффективный асинхронный параллелизм и применимость к схемам с последовательностной логикой с обнаружением критических циклов. Использование асинхронного параллелизма позволяет распределять нагрузку на вычислительные ресурсы более равномерно, чем известные алгоритмы с ранжированием вершин и синхронизацией по уровням.

Алгоритм протестирован на современных цифровых схемах, содержащих десятки тысяч логических элементов. Ускорение алгоритма существенно зависит от вида и размера схем. Оно тем выше, чем больше схема и больше в ней независимых путей. Максимальное достигнутое нами ускорение – 8,9 раз на 16 ядрах. Полученные результаты говорят о перспективности работ по параллелизации статического временного анализа. Возможность приближения к линейному ускорению алгоритмов на данный момент остается открытым вопросом.

Литература

[1] Соловьев Р.А., Глебов А.Л., Гаврилов С.В. Статический временной анализ с обнаружением ложных проводящих путей на основе логических импликаций // Всероссийская научно-техническая конференция "Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС)". Сборник трудов ИППМ РАН. 2006. С. 78-84.

[2] Гаврилов С., Стемпковский А., Глебов А. Методы логического и логико-временного анализа цифровых СБИС / М.: Наука, 2007. 220 с.

[3] Матросова А.Ю., Кудин Д.В., Николаева Е.А. Обнаружение ложных путей в комбинационной схеме // Вестник Томского Государственного Университета, Вычислительная техника и информатика. 2011. №2. С. 99 – 107.

[4] Yen S., Du D., Ghanta S. Efficient Algorithms for Extracting the K Most Critical Paths in Timing Analysis // In Proc. Design Automation Conference (DAC). 1989. P. 649-654.

[5] Tsay R., Lin I. Robin Hood: A System Timing. Verifier for Multi-Phase Level Sensitive Clock Designs // In Proc. Fifth Annual IEEE International IBM. 1992. P. 516–519.

[6] Hassoun S., Sasao T. Logic synthesis and verification / Kluwer Academic Publishers. 2002. V. 654. P. 373-401.

[7] Ирбенек В. С. Временная верификация и оптимизация размещения компонентов предельных по быстродействию ЭВМ / Дис. на соиск. уч. ст. д-ра техн. наук, Институт микропроцессорных вычислительных систем Российской академии наук. Москва, 2001. 193 с.

[8] Lee J., Tang D T., Wong C.K. Timing Analysis Algorithm for Circuits with Level-Sensitive Latches // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 1996. №15. P. 535 – 543.

[9] Burks T. M., Sakallah K. A., Mudge T. N. Critical Paths in Circuits with Level-Sensitive Latches // IEEE Transactions on Very Large Scale Integration (VLSI) systems. 1995. № 3. P. 273-291.

[10] Князев Н.А. Статический временной анализ синхронных цифровых схем на многоядерных ЭВМ // Труды Международной суперкомпьютерной конференции Научный сервис в сети Интернет: экзафлопсное будущее. Новороссийск, 2011. С. 617-623.

[11] Holder A., Christopher D.C., Kalafala K. Prototype for a Large-Scale Static Timing Analyzer running on an IBM Blue Gene // in Proc. Parallel & Distributed Processing, Workshops and Phd Forum (IPDPS) Workshops. 2010. P. 1-8.

[12] Lee J., Tang D.T. An Algorithm for Incremental Timing Analysis // Proceeding DAC '95 Proceedings of the 32nd annual ACM/IEEE Design Automation Conference. 1995. P. 696-701.

[13] Reindes J. Intel Threading Building Blocks / O'Reilly Media, Inc. Sebastopol, 2007. 303 p.