УДК 681.518

Д. Е. Иванов, канд. техн. наук
Ин-т прикладной математики и механики НАН Украины
(Украина, 83114, Донецк, ул. Р.Люксембург, 74,
тел.: (062) 3116795, (067) 2812648,
E-mail: ivanov@iamm.ac.donetsk.ua)

Параллельный алгоритм моделирования цифровых схем с неисправностями для многоядерных систем с общей памятью

Предложен новый параллельный алгоритм моделирования цифровых схем с неисправностями, основанный на одновременном многопоточном моделировании групп неисправностей для каждого входного набора, когда в каждом потоке параллельно по разрядам машинного слова моделируется группа неисправностей. Для ускорения работы дополнительно использовано динамическое разбиение списка неисправностей на такие группы. Приведены результаты машинных экспериментов со схемами ISCAS-89, проведенных на 12-тиядерной рабочей станции.

Запропоновано новий паралельний алгоритм моделювання цифрових схем із пошкодженнями, базований на одночасному багатопоточному моделюванні груп пошкоджень для кожного вхідного набору, коли в кожному потоці паралельно за розрядами машинного слова моделюється група пошкоджень. Для прискорення процесу додатково використано динамічне розбиття списку пошкоджень на такі групи. Наведено результати машинних експериментів зі схемами ISCAS-89, проведених на 12-тиядерній робочій станції.

К л ю ч е в ы е с л о в а: цифровая схема, моделирование неисправностей, параллельные вычисления.

Алгоритмы моделирования устройств с неисправностями входят в число наиболее используемых разработчиками цифровых схем (ЦС). Основное их назначение — определение множества неисправностей, обнаруживаемых заданной входной последовательностью. Результатом работы алгоритмов моделирования являются списки проверенных (непроверенных) неисправностей, а их общим недостатком можно считать то, что для больших схем процесс моделирования может продолжаться несколько десятков часов в связи с большими списками анализируемых неисправностей. Поэтому актуальной остается задача построения быстрых алгоритмов моделирования ЦС с неисправностями. Дополнительным стиму-

лом при этом является развитие алгоритмов построения тестов, основанных на моделировании [1, 2].

Первые попытки ускорения процесса моделирования ЦС связаны с применением аппаратных акселераторов [3]. Развитие параллельных вычислительных систем привело к разработке параллельных алгоритмов моделирования [4, 5]. Наиболее полно построение параллельного алгоритма моделирования ЦС с неисправностью рассмотрено в работе [6]. Алгоритм предназначен для быстрого определения диагностических свойств представленной входной последовательности. В качестве меры принята полнота относительно полного списка одиночных константных неисправностей. Следует заметить, что в случае, когда необходимо восстанавливать поведение схемы при наличии одной либо нескольких неисправностей, данный подход неприменим, поскольку неисправность исключается из рассмотрения в тот момент, когда она обнаружена первый раз.

В настоящее время известны три основные схемы распараллеливания алгоритмов моделирования ЦС с неисправностями.

1. *Разбиение списка неисправностей* [5, 7, 8]. Полный список неисправностей F разбивается на несколько подсписков $F_1, F_2, ..., F_n$, каждый из которых передается на отдельный процессор системы, где выполняется его анализ моделированием на заданной входной последовательности. При такой схеме каждый узел вычислительной среды должен иметь свою копию описания схемы и теста. В настоящее время данный метод получил наибольшее распространение, так как обладает хорошей масштабируемостью при увеличении числа процессоров. Подобные алгоритмы для кластерных систем с распределенной памятью и для многоядерных рабочих станций с общей памятью описаны в работах [9, 10].

2. *Разбиение схемы* [11, 12]. Схема разбивается на несколько подсхем, каждая из которых моделируется на отдельном узле вычислительной системы. Подход применяется для моделирования исправных схем и схем с неисправностями. Основным преимуществом такого подхода является существенное уменьшение требований к затратам памяти, в которой хранится описание схемы, а недостаток заключается в том, что отдельные задачи в процессе работы должны взаимодействовать между собой, что ведет к сложности реализации [13].

3. *Разбиение теста* [14] заключается в том, что входная последовательность T разбивается на ряд подпоследовательностей $T_1, T_2, ..., T_n$, с которыми формируются подзадачи (процессы) для вычислительных узлов. Во время работы процессы взаимодействуют между собой, информируя о том, какие неисправности обнаружены к текущему моменту времени. Если в заданной входной последовательности не известны такты

времени, включающие подпоследовательности (что случается достаточно часто), то данный метод не может быть применен.

Возобновление интереса к параллельным вычислениям и алгоритмам обусловлен широким распространением рабочих станций с многоядерными процессорами [15]. Для разработчиков такие системы классифицируются как параллельные вычислительные системы с общей памятью. Предлагаемый новый параллельный алгоритм моделирования ЦС с неисправностями ориентированный на работу в таких вычислительных системах.

К трем перечисленным выше схемам добавляется еще одна, согласно которой распараллеливанию подвергается моделирование всех неисправных схем на текущем входном наборе. При этом дополнительно используется параллельное по разрядам машинного слова моделирование, а неисправности, обрабатываемые вместе, называются группой.

Моделирование нескольких сформированных групп неисправностей осуществляется параллельно в различных потоках. Проверенные текущим набором неисправности вычеркиваются из списка и не включаются в очередь на моделирование в дальнейшем. При этом фактически при моделировании каждого входного набора реализуется динамическое разбиение списка неисправностей, так как состав групп постоянно меняется на каждом такте модельного времени.

Параллельный по разрядам машинного слова алгоритм моделирования с динамической сортировкой неисправностей. Алгоритм моделирования цифровых устройств с неисправностями, являющийся базовым для параллельной версии, идейно основан на алгоритме PROOFS [16], подробное описание которого приведено в [17]. Основным преимуществом данного алгоритма является параллельное по разрядам машинного слова моделирование неисправностей, что существенно повышает быстродействие. Алгоритм рассчитан на работу с синхронными последовательностными схемами, в которых выделяется комбинационная часть и элементы состояний, представленные *D*-триггерами. Моделирование выполняется по тактам времени итерационным моделированием комбинационных эквивалентов.

Рассмотрим подробнее структуру данного алгоритма, так как ее понимание является существенным для построения параллельной версии. Укрупненный псевдокод 1 алгоритма имеет следующий вид:

```
Моделирование (Схема, СписокНеисправностей, ВходнаяПоследовательность,
ЧислоНаборов)
{
    Инициализация();
    while( НомерНабора < ЧислоНаборов )
```

```
    {
        Набор=ВходнаяПоследовательность[НомерНабора];
        SVI=МоделироватьИсправнуюСхему(Схема, Набор);
        while( в списке есть непроверенные неисправности )
        {
            Группа=СформироватьГруппуНеисправностей(СписокНеисправностей);
            МоделироватьНеисправныеСхемыГруппы(Схема, Набор, Группа);
            ПроверитьТестируемостьНеисправностейВГруппе(SVI, Группа);
            СохранитьСостоянияДляНепроверенныхНеисправностей(SVI, Группа);
        } // конец цикла по неисправностям
    }     // конец цикла по входным наборам
}         //  конец ядра моделирования
```

Основной цикл моделирования по входным наборам включает следующие шаги. Для текущего входного набора происходит моделирование работы исправной схемы, результатом которого является массив SVI значений сигналов на всех линиях. Значения сигналов из данного массива используются далее для определения:

1) активности неисправностей (неактивные в данный момент модельного времени неисправности не включаются в очередь на моделирование);

2) проверяемости неисправностей после их моделирования;

3) элементов состояний неисправных схем, которые следует сохранить для моделирования на следующих входных наборах.

Далее открывается цикл по списку неисправностей. Пока в списке остались неисправности, не рассмотренные на текущем входном наборе, выполняются следующие действия. Сначала формируется группа неисправностей, которые будут моделироваться параллельно в разных разрядах машинного слова. Затем выполняется параллельное по разрядам машинного слова моделирование работы группы неисправных схем. После этого выполняется проверка обнаружимости неисправностей данным входным набором. Проверенная неисправность вычеркивается из списка неисправностей. Если некоторая неисправность не проверилась, то для нее выполняется сохранение состояний триггеров. После этого продолжаются циклы по неисправностям и входным наборам.

Основные особенности рассматриваемого алгоритма следующие.

Динамическое сжатие списка неисправностей. Неисправность, обнаруженная на текущем входном наборе, удаляется из списка неисправностей и, следовательно, не рассматривается в дальнейшем.

Параллельное моделирование неисправностей в разрядах машинного слова динамическим объединением в группы. Поскольку список непрове-

ренных неисправностей постоянно изменяется, изменяется и состав групп неисправностей, формируемых для совместного моделирования.

Сохранение состояний неисправных схем. Необходимо запомнить состояние неисправной схемы, полученное в результате моделирования неисправности в группе. С этой целью для каждой неисправности создается список значений состояний триггеров, в котором сохраняются состояния только тех триггеров, значения которых отличны от значений в исправной схеме.

Параллельный по группам неисправностей алгоритм моделирования является многопоточной версией алгоритма, описанного выше.

Из псевдокода 1 видно, что он имеет два цикла: по входным наборам и по неисправностям в списке. Распараллеливание наибольших фрагментов кода должно обеспечивать лучшую масштабируемость вследствие уменьшения доли последовательного кода. Поэтому распараллеливание следует начинать с самого верхнего цикла. Однако в данном случае это не представляется возможным, так как нельзя выполнять моделирование на следующем входном наборе i 1, не закончив моделирование на текущем наборе i, поскольку не будут определены состояния схем в момент перехода от набора i к набору i 1.

В цикле по неисправностям расположено несколько функций. Наибольшая вычислительная нагрузка приходится на функцию МоделироватьНеисправныеСхемыГруппы. Выполнение моделирования группы неисправностей на текущем входном наборе не влияет на моделирование другой группы на этом же наборе. Поэтому моделирование таких групп представляет собой независимые ветви программы, и можно организовать параллельное выполнение нескольких таких функций.

Основным программным средством реализации параллельных вычислений в современных средах программирования являются потоки. Их парадигму поддерживают все современные языки программирования. Потоки представляют собой фрагменты программного кода, которые выполняются параллельно, т.е. не в предписанном порядке во времени. Их взаимодействие обеспечивают специальные механизмы и процедуры.

Основные операции с потоками следующие: создание, запуск на выполнение, приостановка, ожидание окончания и удаление. Современные процессоры сами определяют, на каком из ядер будет выполняться код потока. Существуют средства привязки потока к вычислительным ядрам, однако это часто замедляет выполнение потока. В параллельной вычислительной среде наибольшее ускорение получается в случае привязки потоков, допускающей их миграцию между ядрами внутри одного сокета либо вычислительного узла [18]. Число потоков, необходимое для параллельного запуска, следует определять экспериментально.

Таким образом, для организации параллельного выполнения функцию МоделироватьНеисправныеСхемыГруппы необходимо реализовать как потоковый класс, который будет содержать программный код соответствующей функции из последовательной реализации. Поскольку такие потоки являются независимыми и не взаимодействуют, нет необходимости синхронизировать их выполнение.

В цикле по неисправностям расположены также несколько функций. Очевидно, что процедуры ПроверитьТестируемостьНеисправностейВГруппе и СохранитьСостоянияДляНепроверенныхНеисправностей легко переносятся в формируемый поток. Функция СформироватьГруппуНеисправностей является подготовительной и не может быть внедрена в поток. Однако это не должно повлиять на производительность, поскольку функция выполняет самые минимальные вычислительные действия: в соответствующий массив заносятся указатели неисправностей, формирующих группу.

Псевдокод 2 такой реализации в терминах потоков имеет следующий вид:

```
ПараллельноеМоделирование1(Схема, СписокНеисправностей,
ВходнаяПоследовательность, ЧислоНаборов, ЧислоПотоков)
{
    Инициализация();
    СоздатьПотокиМоделированияНеисправностей();
    while( НомерНабора < ЧислоНаборов )
    {
        Набор=ВходнаяПоследовательность[НомерНабора];
        SVI=МоделироватьИсправнуюСхему(Схема, Набор);
        while( в списке есть непроверенные неисправности )
        {
            for( НомерПотока=0 ; НомерПотока<ЧислоПотоков ; НомерПотока++ )
            {
              Группа=СформироватьГруппуНеисправностей(СписокНеисправностей);
              ПотокМоделированияНеисправностей[НомерПотока]->Данные(Набор,SVI,
              Группа);
              ПотокМоделированияНеисправностей[НомерПотока]->ЗапуститьПоток();
            }
            // ждать окончания моделирования всех потоков с неисправностями
            for( НомерПотока=0 ; НомерПотока<ЧислоПотоков ; НомерПотока++ )
            {
               ПотокМоделированияНеисправностей[НомерПотока]->ЖдатьОкончания();
            }
        } // конец цикла по неисправностям
```

```
    } // конец цикла по входным наборам
    УдалитьПотокиМоделирования();
}       // конец ядра моделирования
```

Здесь переменная ЧислоПотоков показывает число одновременно выполняемых потоков. Ее значение следует определять экспериментально. Рекомендуется при построении многопоточных приложений выбирать значение, равное числу логических ядер в системе [19].

Массив ПотокМоделированияНеисправностей содержит необходимое число указателей на объект данного класса. В него включен код функций МоделироватьНеисправныеСхемыГруппы, ПроверитьТестируемостьНеисправностей ВГруппе и СохранитьСостоянияДляНепроверенныхНеисправностей из псевдокода 1 алгоритма.

Частое применение процедуры создания (удаления) потоков может существенно уменьшить производительность приложения. Поэтому в начале работы следует создать пул потоков, работа с которыми осуществляется запуском и приостановкой.

Из псевдокода 2 видно, что выполнено распараллеливание фрагмента, в котором осуществляется моделирование работы групп неисправных схем. Перед этим фрагментом выполняется функция моделирования работы исправной схемы. Следовательно, перед параллельным участком кода стоит большой фрагмент последовательного кода, который также желательно выполнить параллельно с остальными. Однако напрямую создать и запустить поток моделирования исправной схемы невозможно, поскольку в ней вычисляются значения на линиях схемы, необходимые для следующих ниже по коду процедур моделирования с неисправностями.

В то же время, распараллеливание данного фрагмента возможно. Для этого создаем дополнительный поток, содержащий код функции МоделироватьИсправнуюСхему. На шаге i моделируем работу исправной схемы на наборе i 1, результаты которой используем для моделирования групп неисправностей на следующей итерации цикла по входным наборам. Перед первым тактом моделирования, т.е. до открытия цикла по входным наборам, следует выполнить моделирование работы исправной схемы. При этом придется создать две копии массива значений сигналов в исправной схеме: SVI —используется для моделирования групп неисправностей на текущем входном наборе, SVINEXT — для предварительного вычисления значений в исправной схеме.

После окончания моделирования работы исправной схемы перед переходом к следующему набору необходимо переписать значения из SVINEXT в SVI, что реализуется простым присваиванием указателя.

Псевдокод 3 такой реализации следующий:

```
ПараллельноеМоделирование2(Схема, СписокНеисправностей,
ВходнаяПоследовательность, ЧислоНаборов, ЧислоПотоков)
{
    Инициализация();
    МоделированиеИсправнойСхемы ->СоздатьПоток();
    СоздатьПотокиМоделированияНеисправностей(ЧислоПотоков);
    // выполнить моделирование для исправной схемы на первом наборе
    МоделированиеИсправнойСхемы->Данные(ВходнаяПоследовательность[0]);
    МоделированиеИсправнойСхемы->ЗапуститьПоток();
    SVI=МоделированиеИсправнойСхемы->ЖдатьОстановки();
    while( ТекущийНабор < ЧислоНаборов )
    {
      Набор=ВходнаяПоследовательность[НомерНабора];
      СледующийНабор=ВходнаяПоследовательность[НомерНабора+1];
      // запускаем поток моделирования исправной схемы для
      // следующего входного набора
      МоделированиеИсправнойСхемы->ПодготовитьДанные(Схема, СледующийНабор);
      МоделированиеИсправнойСхемы->ЗапуститьПоток();
      while( в списке есть непроверенные неисправности )
      {
        for( НомерПотока=0 ; НомерПотока<ЧислоПотоков ; НомерПотока++ )
        {
          Группа=СформироватьГруппуНеисправностей(СписокНеисправностей);
          ПотокиМоделированияНеисправностей[НомерПотока]->Данные(Набор, SVI,
          Группа);
          ПотокиМоделированияНеисправностей[НомерПотока]->ЗапуститьПоток();
        }
        // ждать окончания моделирования всех потоков с неисправностями
        for( НомерПотока=0 ; НомерПотока<ЧислоПотоков ; НомерПотока++ )
        {
          ПотокМоделированияНеисправностей[НомерПотока]->ЖдатьОстановки();
        }
      } // конец цикла по неисправностям
      // ждать окончания моделирования исправной схемы для следующего набора
      SVINEXT=МоделированиеИсправнойСхемы->ЖдатьОстановки();
      // запомнить состояние исправной схемы для моделирования
      // с неисправностями на след.итерации
      SVI=SVINext;
```

```
    } // конец цикла по входным наборам
    УдалитьПотокиМоделирования();
}       // конец ядра моделирования
```

Как видно из псевдокода 3, распараллелены оба главных цикла последовательной версии алгоритма. С учетом дополнительного потока моделирования исправной схемы число параллельных потоков равно ЧислоПотоков +1.

Построенный по такой схеме алгоритм моделирования не относится ни к одной из перечисленных выше схем построения параллельных версий. Наиболее близок он к схеме 1, согласно которой выполняется разбиение теста для моделирования на различных узлах вычислительной системы. Однако, рассматриваемая схема не имеет предварительного разбиения списка неисправностей и привязки подсписков к узлам вычислительной системы. Вместо этого разбиение на группы неисправностей происходит динамически для каждого входного вектора во входной последовательности.

Кроме того, алгоритм рассчитан на многопроцессорную систему с общей памятью и, следовательно, нет необходимости передавать каждому вычислительному ядру описание схемы и формируемые списки неисправностей. Если появится необходимость адаптировать алгоритм к многопроцессорным системам с раздельной памятью, то следует предусмотреть соответствующие процедуры передачи данных и приема результатов моделирования [9].

Экспериментальные данные. Предоставленный компанией Intel доступ к лаборатории MTL (Manycore Testing Lab), позволил дистационно организовать работу с рабочей станцией под управлением операционной системы MS Windows Server 2008 R2. Многоядерная ВС содержала два процессора Intel(R) Xeon CPU X5650 с частотой 2,67 Ггц (в каждом по шесть вычислительных ядер), технология HyperThreading — выключена, объем оперативной памяти 16 Гб. Функция GetSystemInfo() показывала наличие 12-ти вычислительных ядер в системе.

Проведено две серии экспериментов, целью которых было получение ответов на такие вопросы:

1. Следует ли использовать отдельное описание схем для каждого потока моделирования?

2. Насколько хорошо масштабируем предложенный алгоритм?

Для проведения экспериментов использованы контрольные схемы из международного каталога ISCAS-89 [20].

Первая серия экспериментов (предварительная). Известно, что подсистемы памяти в параллельных вычислительных системах являются «уз-

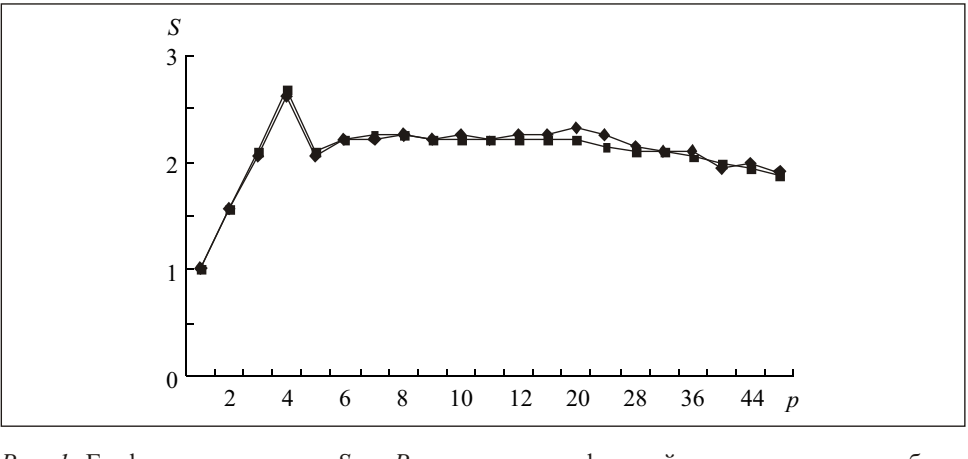

Рис. 1. Графики зависимости S от P для двух модификаций алгоритма: — общие таблицы; ■ — раздельные таблицы

ким» местом, поскольку с их помощью все процессоры должны быть обеспечены необходимыми данными для работы. Наибольший объем памяти при моделировании требуется для описания схемы в виде таблиц [21]. Следует ли в системах рассматриваемого класса для каждого потока моделирования использовать отдельное описание в виде таблиц или достаточно одного общего описания?

Для получения ответа на данный вопрос программно реализованы две модификации алгоритма. В первой модификации имелась одна копия таблиц описания схемы, а во второй — до начала процесса моделирования было создано необходимое число копий данных таблиц (по числу потоков моделирования), указатели на которые передавались потокам в качестве аргументов.

Эксперименты были проведены для схемы средней размерности s9234 и состояли в запуске программы моделирования с изменяемым числом одновременно выполняемых потоков. На рис. 1 приведены графики коэффициента ускорения S работы алгоритма в зависимости от числа параллельных потоков P для обеих модификаций. Из рис. 1 видно, что практически не имеет значения, какие таблицы описания схем использованы: общие или раздельные. Поэтому дальнейшая разработка алгоритма выполнялась для модификации с общими таблицами.

Вторая серия экспериментов. Были исследованы свойства масштабируемости параллельного алгоритма моделирования по группам неисправностей. Число одновременно выполняемых потоков изменялось от одного до 18. Максимальное число потоков выбрано больше числа ядер системы,

чтобы можно было точно определить экстремум в графиках ускорения. Численные результаты экспериментов приведены в таблице.

Важными характеристиками параллельных алгоритмов являются ускорение S, эффективность использования ядер E и доля f последовательного кода [22].

Ускорение при параллельной реализации алгоритма для P процессоров (потоков), определяем по формуле

$$S_p(n) \quad \frac{T_1(n)}{T_p(n)},$$

где n — параметр вычислительной сложности алгоритма; $T_i(n)$ — время выполнения параллельного алгоритма на системе с i процессорами.

Эффективность использования процессоров (ядер) при параллельной реализации алгоритма рассчитываем по формуле

$$E_p(n) \quad \frac{T_1(n)}{p\,T_p(n)} \quad \frac{S_p(n)}{p}.$$

Доля последовательного кода (последовательных вычислений) f является оценкой свойств алгоритма к распараллеливанию и определяется по закону Амдала [22]:

$$f \quad \frac{p/S_p \quad 1}{p \quad 1}.$$

В таблице приведены вычисленные значения указанных параметров. Следует заметить, что значения S несколько занижены относительно последовательной версии алгоритма. Это связано с тем, что когда в качестве параметра указан один поток моделирования с неисправностями, в действительности выполняются два потока: поток моделирования исправной схемы для следующего входного набора и поток моделирования группы неисправностей для текущего набора.

Для точного замера времени работы однопотокового приложения необходимо выполнить последовательный запуск этих потоков. Однако такой возможности не было в связи с ограниченным временем доступа к MTL. Поэтому оценка была выполнена на компьютере с четырехядерным процессором Intel Core2 Quad Q6600. Указанные изменения внесены в программный код и определено значение S, достигаемое при выделении в отдельный поток моделирования поведения исправной схемы. В экспериментах оно составляло от двух до четырех процентов. Следовательно, в таких пределах находится и точность данных, приведенных в таблице, что является приемлемым для качественной оценки.

Анализ приведенных числовых значений позволяет сделать вывод о том, что наибольшее ускорение для всех схем достигается, когда параллельно выполняется 11 потоков моделирования групп неисправностей. При этом реальное число параллельных потоков равно 12, т.е. 11 потоков моделирования групп неисправностей плюс поток моделирования исправной схемы, что соответствует числу вычислительных ядер в системе.

Графики зависимости коэффициента ускорения от числа одновременных вычислительных потоков для случаев наибольшего, наименьшего и усредненного ускорения приведены на рис. 2.

Следует заметить, что для всех схем наименьшее значение S не превышает 4,61. При этом разброс значений данного параметра составляет от 4,61 до 6,72. Это качественно отличает полученные результаты от результатов работы [6], в которой разброс значений этого параметра составляет от 1,10 до 7,31 на восьмипроцессорной системе iPSC/860.

Число потоков	Характеристики параллельного алгоритма моделирования ЦС с неисправностями														
	s9234			s35932			s38417			s38584			s38584_1		
	S	*E*	*f*	*S*	*E*	*f*	*S*	*E*	*f*	*S*	*E*	*f*	*S*	*E*	*f*
1	**98**	0,08	1	**619**	0,08	1	**2441**	0,08	1	**2792**	0,08	1	**1512**	0,08	1
2	1,78	0,15	0,52	1,96	0,16	0,47	2,04	0,17	0,44	1,92	0,16	0,48	1,38	0,11	0,70
3	2,51	0,21	0,34	2,88	0,24	0,29	2,90	0,24	0,29	2,79	0,23	0,30	2,00	0,17	0,45
4	3,16	0,26	0,25	3,73	0,31	0,20	3,86	0,32	0,19	3,66	0,31	0,21	2,62	0,22	0,32
5	3,38	0,28	0,23	4,30	0,36	0,16	4,40	0,37	0,16	4,20	0,35	0,17	3,05	0,25	0,27
6	3,06	0,26	0,27	3,92	0,33	0,19	3,89	0,32	0,19	3,81	0,32	0,20	2,70	0,23	0,31
7	3,38	0,28	0,24	4,30	0,36	0,16	4,50	0,38	0,15	4,28	0,36	0,16	3,12	0,26	0,26
8	3,77	0,31	0,20	4,84	0,40	0,13	4,83	0,40	0,13	4,84	0,40	0,13	3,60	0,30	0,21
9	4,08	0,34	0,18	5,29	0,44	0,12	5,35	0,45	0,11	5,25	0,44	0,12	3,99	0,33	0,18
10	4,26	0,36	0,17	5,68	0,47	0,10	5,77	0,48	0,10	5,67	0,47	0,10	4,26	0,35	0,17
11	4,67	0,39	0,14	6,19	0,52	0,08	6,72	0,56	0,07	6,24	0,52	0,08	4,61	0,38	0,15
12	4,45	0,37	0,15	5,07	0,42	0,12	5,42	0,45	0,11	4,99	0,41	0,13	3,34	0,28	0,24
13	3,92	0,33	0,19	4,95	0,41	0,13	5,31	0,44	0,11	5,09	0,42	0,12	3,54	0,30	0,22
14	3,92	0,33	0,19	4,73	0,39	0,14	5,55	0,46	0,11	5,19	0,43	0,12	3,61	0,30	0,21
15	3,92	0,33	0,19	4,45	0,37	0,15	5,05	0,42	0,12	5,09	0,42	0,16	3,48	0,29	0,22
16	3,06	0,26	0,27	4,33	0,36	0,16	4,69	0,42	0,14	4,40	0,37	0,16	3,38	0,28	0,23
17	2,97	0,25	0,28	4,27	0,36	0,16	4,71	0,39	0,14	4,38	0,36	0,16	3,36	0,28	0,23
18	2,80	0,23	0,30	4,21	0,35	0,17	4,66	0,39	0,14	4,42	0,37	0,16	3,35	0,28	0,23

Примечание. Полужирным шрифтом указано время моделирования T в секундах; остальные значения S, E и f — безразмерные величины.

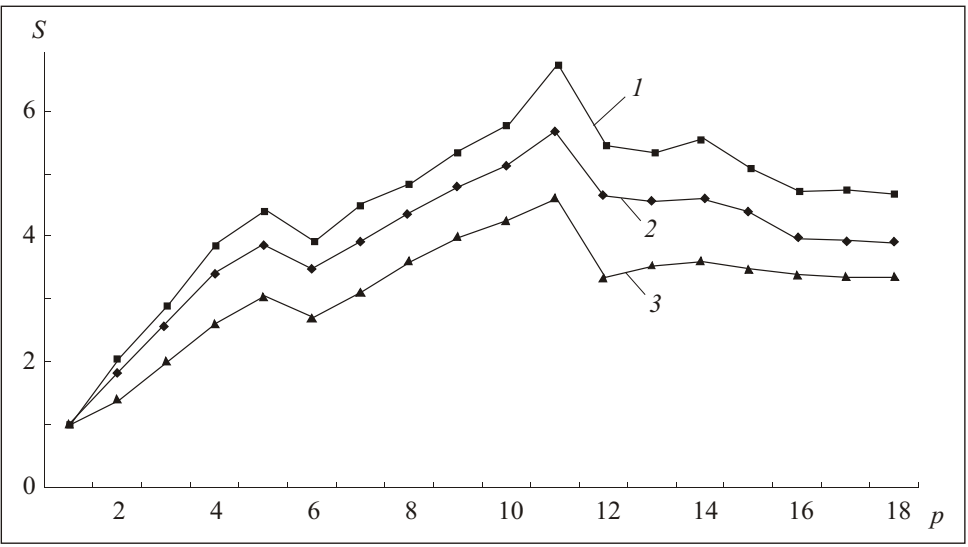

Рис. 2. Зависимости наилучших (*1*), усредненных (*2*) и наихудших (*3*) значений *S* от числа потоков моделирования с неисправностями

Выводы. Предложенный новый параллельный алгоритм моделирования цифровых устройств с неисправностями заключается в распараллеливании моделирования групп неисправностей для каждого входного набора. Для уменьшения доли последовательного кода дополнительно организуется поток моделирования работы исправной схемы для следующего входного набора.

Результаты машинных экспериментов свидетельствую о хорошей масштабируемости параллельной версии алгоритма.

Данный параллельный алгоритм может быть применен к любому типу неисправностей, для которых допустимо параллельное по разрядам машинного слова моделирование. Если свойство параллельного по разрядам машинного слова моделирования не принципиально, то алгоритм можно использовать для любых моделируемых типов неисправностей.

В дальнейшем возможно построение алгоритма моделирования, который позволит объединить свойства описанного алгоритма и статическое разбиение списка неисправностей.

Авторы выражают благодарность компании Intel , а также ее подразделению Intel Software Network за предоставленную возможность доступа к 12-тиядерной рабочей станции лаборатории Manycore Testing Lab, а также личную благодарность Майку Пирсу и Питеру Гинсбику за оказанную техническую поддержу во время сессии доступа и после нее.

A new algorithm for parallel many-threaded fault simulation of digital circuits is proposed. It is based on the concurrent many-threaded simulation of the groups of faults for each input vector. Each group of faults is simulated in bit-parallel way. The dynamic fault partitioning for forming such groups is used to speed-up the algorithm. The results of computational experiments on ISCAS-89 benchmarks circuits are reported, which are obtained on the 12-core workstation.

1. *Corno F., Prinetto P., Rebaudengo M., Sonza Reorda M.* GATTO: a Genetic Algorithm for Automatic Test Pattern Generation for Large Synchronous Sequential Circuits // IEEE Transactions on Computer-Aided Design, August. — 1996. — **15**, № 8. — P. 943—951
2. *Иванов Д. Е.* Генетические алгоритмы построения идентифицирующих последовательностей для цифровых схем с памятью // Наук. праці Донецького національного технічного університету. Серія «Обчислювальна техніка та автоматизація». — Донецьк : ДонНТУ. — 2008. — Вип. 14 (129). — С. 97—106.
3. *Blank T. A.* Survey of Hardware Accelerators Used in Computer Aided Design // IEEE Design and Test. — 1984. — P. 21—39.
4. *Mueller-Thuns R. B., Saab D. G., Damiano R. F., Abraham J. A.* VLSI Logic and Fault Simulation on General-Purpose Parallel Computers // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. — 1993. — №12 (3). — P. 446—460.
5. *Duba P. A., Roy R. K., Abraham J. A., Rogers W. A.* Fault Simulation in a Distributed Environment // Proc. of 25th Design Automation Conf. — 1988. — P. 686—691.
6. *Parker S., Banerjee P., Patel J.* A parallel algorithm for fault simulation based on PROOFS // Proc. IEEE Int. Conf. Computer Design. — 1995. — P. 616—621.
7. *Krishnaswamy D., Banerjee P., Rudnick E. M., Patel J. H.* Asynchronous parallel algorithms for test set partitioned fault simulation // ACM SIGSIM Simulation Digest. — 1997. — **27**, № 1. — P. 30—37.
8. *Markas T., Royals M., Kanopoulos N.* On distributed fault simulation // IEEE Computer. — 1990. — **7.** — P. 40—52.
9. *Иванов Д. Е., Скобцов Ю. А., Эль-Хатиб А. И.* Распределенное параллельное моделирование цифровых схем с неисправностями // Наук. праці Донецького національного технічного університету. Серія «Обчислювальна техніка та автоматизація». — Донецьк : ДонНТУ. — 2006. — Вип. 107. — С. 128—134.
10. *Ivanov D. E.* Parallel fault simulation on multi-core processors // Радіоелектронні і комп'ютерні системи. — 2009. — № 6 (40). — С. 109—112.
11. *Ghosh S.* A distributed algorithm for fault simulation of combinatorial and asynchronous sequential digital designs, utilizing circuit partitioning, on loosely coupled parallel processors // Microelectronic Reliability. — 1995. — № 35 (6). — P. 947—967.
12. *Muller-Thuns R. B., Saab D. G., Damiano R. F., Abraham J. A.* Abraham Portable parallel logic and fault simulation // Digest of paper, International Conf. on Computer Aided Design. — Santa Clara, USA. — 1989. — P. 506—509.
13. *Ладыженский Ю. В., Попов Ю. В.* Программная система для исследования протоколов синхронизации при распределенном событийном логическом моделировании // Наук. праці Донецького національного технічного університету. Серія «Обчислювальна техніка та автоматизація». — Донецьк : ДонНТУ. — 2004. — № 74. — С. 201—209.
14. *Ravikumar C. P., Jain V., Dod A.* Distributed Fault Simulation Algorithms on Parallel Virtual Machine // VLSI Design. — 2001. — **12**, №1. — P. 81—99.
15. *Wirt R.* Intel® Software Insight. Multi-core Capability. — USA: Intel Corporation, 2005. — July. — 11 p.
16. *Niermann T. M., Cheng W. T., Patel J. H.* PROOFS: A Fast, Memory-Efficient Sequential Circuits Fault Simulator // IEEE Trans. CAD. — 1992. — P. 198—207.

17. *Иванов Д. Е., Скобцов Ю. А.* Параллельное моделирование неисправностей для последовательностных схем // Искусственный интеллект. — 1999. — № 1. — С. 44—50.
18. *Копысов С. П., Новиков А. К., Тонков Л. Е., Береснев Д. Е.* Методы привязки параллельных процессов и потоков к многоядерным узлам вычислительных систем // Вест. Удмуртского университета. — 2010. — № 1. — С. 123—132.
19. *Gillespie M.* Масштабирование программных архитектур для многоядерных вычислительных систем будущего // Intel® Software Network. — http: software.intel.com/ru-ru/articles/scaling-software-architectures-for-the-future-of-multi- core- computing/.
20. *Brgles F., Bryan D., Kozminski K.* Combinational profiles of sequential benchmark circuits // Intern. symposium of circuits and systems, ISCAS—89. — 1989. — P. 1929—1934.
21. *Барашко А. С., Скобцов Ю. А., Сперанский Д. В.* Моделирование и тестирование дискретных устройств. — Киев : Наук. думка, 1992. — 288 с.
22. *Гергель В. П., Стронгин Р. Г.* Основы параллельных вычислений для многопроцессорных вычислительных систем. Учеб. пособие / Нижний Новгород: Изд-во ННГУ им. Н. И. Лобачевского, 2003. — 184 с.

Поступила 01.09.10

ИВАНОВ Дмитрий Евгеньевич, канд. техн. наук, ст. науч. сотр. Ин-та прикладной математики и механики НАН Украины (г. Донецк). В 1995 г. окончил Донецкий государственный университет. Область научных исследований — техническая диагностика, эволюционные вычисления, генетические алгоритмы, моделирование и тестирование цифровых устройств.