УДК 681.326

С. А. ЗАЙЧЕНКО,
А. Н. ПАРФЕНТИЙ,
Х. КТЕЙМАН

УСКОРЕНИЕ ОПЕРАЦИЙ НАД МНОЖЕСТВАМИ В ДЕДУКТИВНОМ МЕТОДЕ МОДЕЛИРОВАНИЯ НЕИСПРАВНОСТЕЙ

Виконано аналіз структур даних і алгоритмів при виконанні операцій теорії множин над списками дефектів в дедуктивному методі моделювання несправностей цифрових систем. Запропоновано 4 типа структур даних та процедури їх обробки, що забезпечують максимальну швидкодію базових операцій, необхідних для ефективної програмної реалізації метода.

This paper analyzes data structures and algorithms for performing set theory operations upon defects lists within deductive fault simulation method of digital systems. Suggested 4 types of data structures and calculation procedures, which provide maximum performance for base operations required for effective software implementation of the method.

Постановка проблемы. Актуальность работы определяется необходимостью существенного повышения быстродействия средств моделирования неисправностей и автоматической генерации тестов (ATPG) [1] для сложных цифровых систем, имплементируемых в кристаллы программируемой логики.

Более 50% существующих систем ATPG [1 – 4] используют дедуктивный метод моделирования (*deductive stuck-at fault simulation*), с помощью которого можно получить таблицу покрытия неисправностей, проверяемых тестом. Анализ распределения времени вычислительного цикла обработки тест-вектора в дедуктивном методе показывает, что около 70% времени отводится на выполнение теоретико-множественных операций над списками неисправностей: объединение, пересечение и дополнение (вычитание). Поэтому быстродействие программной реализации дедуктивного метода в большей степени зависит от эффективности выполнения операций теории множеств.

Для программной реализации операций над множествами или списками используются структуры данных и алгоритмы, эффективность которых варьируется в зависимости от количества обрабатываемых элементов. Что касается дедуктивного метода моделирования неисправностей, то здесь особенность вычислений связана с широким диапазоном размеров множеств, участвующих в операциях. Поэтому ни одна из общепринятых в программировании структур данных не обеспечивает дедуктивному методу приемлемой производительности выполнения теоретико-множественных операций.

Цель исследования – анализ и выбор оптимальных структур данных и алгоритмов выполнения теоретико-множественных операций в дедуктивном методе моделирования неисправностей с точки зрения быстродействия и затрат памяти.

К задачам исследования относятся: 1) анализ классических структур данных, применяемых в дискретной математике [5, 6] и программировании [7 – 10] для реализации операций теории множеств; 2) разработка стратегии вычислений, обеспечивающей оптимальные затраты памяти и максимальное быстродействие для каждой цифровой системы при выполнении операций над списками неисправностей; 3) оценка эффективности разработанной стратегии.

Анализ классических структур данных для множеств. Теоретико-множественные операции используются при решении многих математических задач, в том числе связанных с рассматриваемой тематикой моделирования неисправностей. Для хранения элементов и реализации множественных операций существует несколько часто применяемых в дискретной математике и программировании структур данных:

– характеристические векторы (*characteristic vectors*);
– связные списки (*linked lists*);
– бинарные деревья (*binary trees*);
– хэш-таблицы (*hash-tables*).

Перечисленные структуры данных являются базовыми при решении широкого класса проблем, а их быстродействие и затраты памяти обусловлены внутренним содержанием.

Характеристические векторы**.** Характеристический вектор [1, 5, 6] – структура данных для хранения подмножеств универсума U с конечным числом элементов n, представляющая собой двоичное слово длиной n бит, в котором единица в i-м бите интерпретируется как принадлежность i-го элемента множеству U, а ноль – отсутствие такой принадлежности.

Например, для представления множества, содержащего четыре элемента {2, 4, 7, 9} в универсуме из 10 элементов, характеристический вектор будет содержать четыре единицы и шесть нулей в индексах, соответствующих номерам элементов минус один (рис. 1):

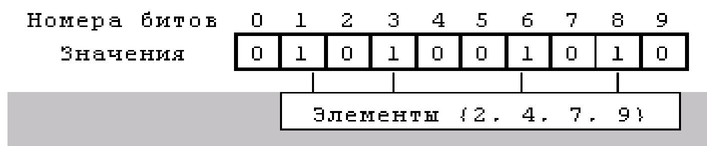

Рис. 1. Представление множеств в виде характеристических векторов

Главное достоинство использования характеристических векторов заключается в простоте программной реализации теоретико-множественных вычислений, сводящихся к следующим логическим операциям:

$$W_{A\cup B} = W_A \vee W_B\,;\; W_{A\cap B} = W_A \cdot W_B\,;\; W_{A-B} = W_A \cdot \overline{W_B}\,, \qquad (1)$$

где $W_{A\cup B}$, $W_{A\cap B}$, W_{A-B} – характеристические векторы результатов объединения, пересечения и разности соответственно, W_A, W_B – характеристические векторы множеств-операндов. Из (1) очевидно, что такая реализация операций теории множеств характеризуется линейной сложностью, зависящей от длины характеристических векторов:

$$\theta_{A\cup B}(n) = t_{OR} \cdot n\,; \quad \theta_{A\cap B}(n) = t_{AND} \cdot n\,; \quad \theta_{A-B}(n) = (t_{AND} + t_{NOT}) \cdot n\,, \qquad (2)$$

где $\theta_{A\cup B}$, $\theta_{A\cap B}$, θ_{A-B} – сложность операций объединения, пересечения и вычитания соответственно, t_{OR}, t_{AND}, t_{NOT} – времена выполнения логических операций ИЛИ, И, НЕ над аргументами длиной в 1 бит.

Основным недостатком использования характеристических векторов при моделировании неисправностей является квадратичная зависимость размера необходимой памяти от количества линий в устройстве. Уже для моделирования схемы из 10 тысяч вентилей, согласно (2), необходимо выделить 100 миллионов бит. Такое потребление памяти делает характеристические векторы практически неприменимыми для обработки схем, имеющих сотни тысяч и миллионы вентилей. Другой существенный недостаток характеристических векторов состоит в том, что значительная часть памяти не используется в вычислениях, поскольку доля активных элементов в схеме редко превышает 20%.

Связные списки. Связный список [7 – 10] – структура взаимосвязанных блоков информации, содержащих хранимые значения и адреса соседних блоков. Для такого представления множеств объем памяти пропорционален количеству элементов. Осуществлять теоретико-множественные операции на основе связных списков практично, если хранить элементы в упорядоченном виде. При этом не требуется затрачивать время на поиск элементов при сравнении двух множеств.

Для выполнения операции объединения с применением упорядоченных списков используется следующая процедура:

$$\begin{cases} A_i < B_t \rightarrow C_j = A_i;\, j = j+1;\; i = i+1; \\ A_i > B_t \rightarrow C_j = B_t;\, j = j+1;\; t = t+1; \\ A_i = B_t \rightarrow C_j = B_t;\, j = j+1;\; i = i+1;\; t = t+1, \end{cases} \qquad (3)$$

где A, B – операнды; C – результат; i, j, t – номера текущих элементов множеств A, B и C соответственно.

Для выполнения операции объединения необходимо сравнивать элементы списков A и B с последующим инкрементом индекса в операнде, где текущий элемент меньше, следует записывать в результат C. Если

значения элементов двух списков равны, один из них следует записать в C, после чего инкрементировать индексы обоих списков. После выполнения процедуры (3) список C также будет упорядоченным.

Аналогичным образом реализуются операции пересечения (4) и вычитания (5) множеств:

$$\begin{cases} A_i > B_t \to t = t+1; \\ A_i < B_t \to i = i+1; \\ A_i = B_t \to C_j = B_t;\ j = j+1;\ i = i+1;\ t = t+1; \end{cases} \tag{4}$$

$$\begin{cases} A_i > B_t \to t = t+1; \\ A_i = B_t \to i = i+1;\ t = t+1; \\ A_i < B_t \to C_j = B_t;\ j = j+1;\ i = i+1. \end{cases} \tag{5}$$

Процедуры (3) – (5) характеризуются линейной сложностью относительно суммы количества элементов обоих списков:

$$\begin{aligned} &\theta_{A\cup B}(n_1,n_2) = 2\cdot t_{cmp}\cdot \min\ (n_1,n_2) + t_{asg}\cdot(n_1+n_2) + 2\cdot t_{inc}\cdot(n_1+n_2); \\ &\theta_{A\cap B}(n_1,n_2) = (2\cdot t_{cmp} + t_{asg})\cdot \min\ (n_1,n_2) + t_{inc}\cdot(n_1+n_2+\min\ (n_1,n_2)); \\ &\theta_{A-B}(n_1,n_2) = 2\cdot t_{cmp}\cdot \min\ (n_1,n_2) + t_{asg}\cdot n_1 + t_{inc}\cdot(2\cdot n_1+n_2), \end{aligned} \tag{6}$$

где n_1, n_2 – количество элементов в операндах, t_{cmp} – время сравнения двух элементов, t_{asg} – время записи элемента в связных список, t_{inc} – время инкремента индекса.

Бинарные деревья. Бинарное дерево [5 – 6] – ациклический граф представления отношений между элементами, где каждая вершина имеет одну входящую и две исходящие дуги. При этом вес текущей вершины всегда больше веса левого преемника и меньше правого. Для такого представления множеств объем памяти пропорционален числу элементов, что эквивалентно связным спискам. Бинарное дерево называется сбалансированным, если организация его вершин характеризуется приблизительным равенством общего числа левых и правых поддеревьев. Сбалансированное бинарное дерево обеспечивает логарифмическую вычислительную сложность поиска элемента, не зависящую от размера множества сложность вставки элемента в найденной позиции. В несбалансированном состоянии дерево поиска перестает быть эффективным, поскольку для определения позиции элемента требуется большее число операций сравнения.

Хэш-таблицы**.** Хэш-таблица [7 – 9] – структура данных для хранения множеств в виде таблицы элементов, позиция которых определяется хэш-функцией. Хэш-коды интерпретируются как индексы ячеек таблицы, что позволяет быстрым вычислением определить местоположение элемента. Операции поиска и добавления элемента в хэш-таблицу характеризуются амортизированной константной сложностью [7 – 9]. Это означает, что в особых ситуациях операция может оказаться более дорогой. Для хэш-таблиц

такими ситуациями являются коллизии и повторное хэширование. Для уменьшения вероятности возникновения коллизий необходимо увеличивать количество ячеек, что вызывает низко производительную операцию повторного хэширования хранимых элементов. Это означает пересчет индексов с учетом нового размера таблицы и переразмещение элементов.

Реализация пересечения, объединения и разности множеств характеризуется амортизированной линейной сложностью:

$$\begin{gathered}\theta_{A \cap B}(n_1, n_2) = \theta_{A-B}(n_1, n_2) = n_1 \cdot \theta_S(n_2) = n_1 \cdot \theta^+(const); \\ \theta_{A \cup B}(n_1, n_2) = n_1 + n_2 \cdot \theta_S(n_1) = n_1 + n_2 \cdot \theta^+(const),\end{gathered} \tag{7}$$

где $\theta^+(const)$ – амортизированная константная сложность.

Хэш-таблицы намного эффективнее бинарных деревьев для множеств с большим количеством элементов, однако для очень малых множеств (до 10 элементов) хэш-таблицы уступают всем другим структурам данных по эффективности выполнения операций.

Объем памяти при применении хэш-таблиц зависит от количества ячеек, которое, как минимум, вдвое должно превышать количество хранимых элементов, что необходимо для организации эффективного хэширования без частых коллизий. Это больше, чем для бинарных деревьев, но значительно меньше, чем для характеристических векторов.

Разработка эффективной стратегии обработки множеств неисправностей. Использование только одной из перечисленных структур данных для представления списков дефектов в дедуктивном методе не обеспечивает производительности, приемлемой для обработки сверхбольших цифровых систем. Для создания быстродействующей программы моделирования предлагается FLP-стратегия вычислений, основанная на пуле списков неисправностей (*fault list pool*). Ее сущность заключается в централизованном управлении программными объектами, реализующими представление списков неисправностей. Пул автоматически контролирует создание, хранение, выделение и освобождение объектов-списков, а также выбор наиболее производительного объекта для конкретной множественной операции. Пул списков функционирует следующим образом:

1. Осуществляет учет созданных, занятых и освобожденных объектов-списков, разделяя объекты по диапазонам размеров множеств.

2. Перед выполнением множественной операции дедуктивный симулятор запрашивает в пуле объект-список, наиболее эффективный в рассматриваемый момент времени.

В зависимости от размеров операндов и выбранной операции, пул оценивает максимально возможное количество элементов результирующего множества:

$$n_{A \cap B} \le \min(n_A, n_B); \quad n_{A \cup B} \le n_A + n_B; \;\; n_{A-B} \le n_A, \tag{8}$$

где n_A, n_B – размеры множеств операндов, $n_{A \cap B}$, $n_{A \cup B}$, n_{A-B} – размеры результатов пересечения, объединения и разности соответственно.

4. Исходя из подсчитанной оценки размера результирующего множества пул ищет подходящий объект-список, используя алгоритм, представленный на рис 2:

Рис. 2. Алгоритм выбора оптимального объекта-списка

5. Выбранный объект предоставляется дедуктивному симулятору для выполнения множественной операции.

6. Если дедуктивный симулятор более не нуждается в списке неисправностей, объект-список возвращается в пул, после чего он может быть повторно использован в других операциях.

Реализация теоретико-множественных операций с применением пула устраняет описанные недостатки классических структур данных:

1. В соответствии с мощностью множества пулом создаются наиболее производительные объекты для списков. Это позволяет использовать структуры данных при самом эффективном количестве элементов, обеспечивая максимальную производительность относительно других структур данных.

2. Предварительное прогнозирование размеров множеств-результатов снижает время, затрачиваемое на реорганизацию структур данных, повышая быстродействие выполнения операций, а также способствует рациональному потреблению памяти.

Недостатком предложенной стратегии вычислений является невозможность одновременного применения неупорядоченных и упорядоченных структур данных. В частности, хэш-таблицы хранят свои элементы в непредсказуемом порядке, а другие – в упорядоченном. Если множества-операнды упорядочены, а множество-результат таким свойством не обладает, вычисления не вызывают осложнений. Однако когда множество-

аргумент является неупорядоченным, множество-результат может быть только неупорядоченным.

Поскольку использование пула не дает резких колебаний количества хранимых элементов, открывается возможность применения простейшей структуры данных – массива фиксированного размера (*fixed-size arrays*). Без пула его использование бессмысленно, поскольку размеры множества могут значительно изменяться в процессе моделирования. Для реализации операций теории множеств на основе массивов можно применять алгоритмы, используемые для связных списков.

В отличие от связных списков, достоинством массивов является возможность быстрого добавления одиночных элементов, что следует производить одновременно с множественными операциями:

1. В каждом из массивов-операндов при помощи двоичного алгоритма поиска (binary search) [6 – 8] установить ожидаемую позицию вставки одиночного элемента.

2. Массивы-операнды требуется условно разбить на две части – до найденной позиции вставки и после нее.

3. Произвести множественную операцию на первых частях операндов.

4. В конец результирующего массива добавить одиночный элемент.

5. Произвести множественную операцию на вторых частях операндов.

Недостаток использования массивов в пуле – невозможность их эффективного сочетания с другими структурами данных. Тем не менее, быстродействие множественных операций с применением массивов эквивалентно наиболее быстрым связным спискам, а время внесения одиночных элементов превышает даже хэш-таблицы. Помимо производительности, массивы требуют для хранения значительно меньший объем памяти. На рис. 3 приведен график производительности системы дедуктивного моделирования с применением различных стратегий обработки множеств. Результаты подтверждают эффективность разработанной стратегии вычислений. Результаты экспериментов подтверждают существенное превосходство (в 3 – 4 раза) разработанной FLP-стратегии теоретико-множественных вычислений по сравнению с наилучшими широко используемыми в программировании подходами.

Выводы. Предложенная FLP-стратегия программной реализации теоретико-множественных операций позволила создать быстродействующую систему дедуктивного моделирования неисправностей для оценки качества синтезируемых тестов.

Основными научными и практическими результатами исследования являются:

– оценка свойств классических структур данных для реализации теоретико-множественных операций в целях моделирования неисправностей сложных цифровых систем;

– выбор структур данных для выполнения теоретико-множественных операций и разработка FLP-стратегии обработки множеств неисправностей, обеспечивающих максимальную производительность при минимальных затратах памяти;

– объединение эффективности теоретико-множественных операций над списками неисправностей с внесением в список одиночного элемента;

– быстродействующая программная реализация дедуктивного метода моделирования, приемлемая для обработки современных сверхбольших цифровых систем на кристаллах, насчитывающих миллионы логических вентилей.

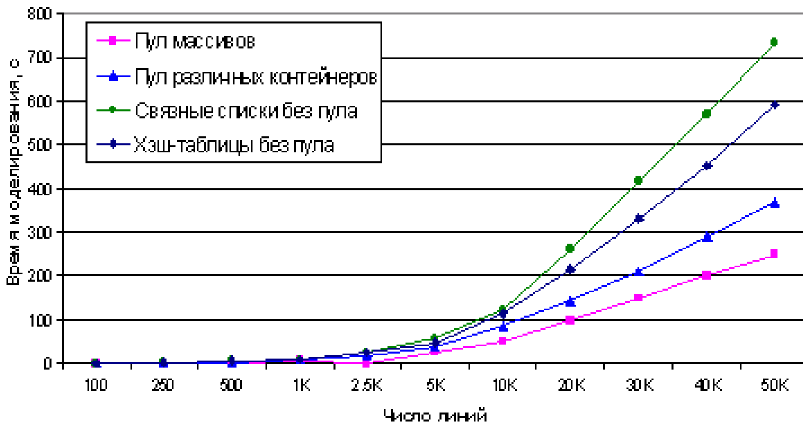

Рис. 3. Производительность разработанной стратегии вычислений

Список литературы: **1.** *Abramovici M., Breuer M.A. and Friedman A.D.* Digital systems testing and testable design. – Computer Science Press, 1998. – 652 p. **2.** *Armstrong D. B.* Deductive Method for Simulating Faults in Logic Circuits // IEEE Transactions on computers. – 1972. – Vol. 1. – C. 21. – P. 464–471. **3.** *Levendel Y.H., Menon P.R.* Comparison of fault simulation methods – Treatment of unknown signal values // Journal of digital systems. – 1980. – Vol. 4. – P. 443–459. **4.** *Hahanov V., Krivoulya G., Hahanova I., Melnikova O., Obrizan V.* High Performance Fault Simulation for Digital Systems // Proceedings of the Second IEEE International Workshop on Intelligent Data Acquisition and advanced Computing Systems: Technology and Application. – Lviv, 2003. – P. 390–395. **5.** *Rossen K.* Discrete Mathematics and its Applications. – McGraw Hill, 2003. – 824 p. **6.** *Горбатов В. А.* Основы дискретной математики. – М.: Высш. шк., 1986. – 311 с. **7.** *Stroustrup B.* The C++ programming Language, 3rd edition. – Addison-Wesley, 2000. – 990 p. **8**. *Eckel B.* Thinking in C++, 2nd edition. – Vol. 2: standard libraries and advanced topics. – Prentice Hall, 2000. – 592 p. **9.** *Meyers S.* “Effective and More Effective C++”. – Addison-Wesley, 2001. – 530 p. **10.** International Standard: Programming Languages – C++, ISO/IEC/ANSI 14882:1998. International Standardization Organization, International Electrotechnical Comission. – American National Standards Insitute, 1998. – 748 p.

Поступила в редакцию 29.09.2004