УДК 519.854.6

А.В. Борисевич
Севастопольский национальный технический университет
ул. Университетская 33, г. Севастополь, Украина, 99053
E-mail: alex.borysevych@gmail.com

ЭФФЕКТИВНАЯ АППАРАТНАЯ РЕАЛИЗАЦИЯ ГЕНЕТИЧЕСКОГО АЛГОРИТМА COMPACT GA ДЛЯ ПОИСКА ЭКСТРЕМУМА ФУНКЦИЙ

Предложена сверхбыстродействующая аппаратная реализация генетического алгоритма для поиска экстремума функций многих переменных, оптимальная по нормированному отношению стоимости к производительности. Рассмотрены различные варианты реализации предложенного подхода на больших программируемых интегральных схемах (FPGA). Даны оценки сложности соответствующих схем.

Введение. Численная оптимизация функций большого числа аргументов чрезвычайно актуальна во многих практических приложениях. Отдельный класс составляют задачи управления на основе процедуры численной оптимизации, периодически применяемой к модели динамического процесса для вычисления траекторий управляющих воздействий с учетом текущего состояния системы (так называемое управление на основе предсказания – MPC [1]). Определенные задачи синтеза и диагностики дискретных систем также ведут к решению задач оптимизации функций от большого числа переменных.

Для решения этих и многих других задач может быть применен генетический алгоритм – метод стохастического поиска экстремума функции, отличающийся большой гибкостью (возможностью применения к неявно заданным, дискретным и разрывным функциям), высокой производительностью и не требующий информации о производных оптимизируемой функции. Генетический алгоритм является эволюционным алгоритмом, т.е. алгоритмом, построенным на прямых аналогиях с процессом эволюции в биологических системах. В целом, генетическому программированию посвящено огромное количество работ. В частности в [1] генетический алгоритм применен к задаче управления на основе предсказания. Генетический алгоритм также является эффективным инструментом построения проверяющих и диагностических тестов для дискретных схем [2].

В условиях использования нелинейных и вероятностных моделей, а также возрастающей вычислительной сложности задач оптимизации, создание функционально-ориентированных аппаратных систем для решения задач оптимизации представляет собой естественный и эффективный подход. В работе [3] была предложена версия генетического алгоритма (Compact GA), реализация которой отличается чрезвычайно малыми затратами памяти и высокой производительностью, не уступающей традиционным реализациям генетических алгоритмов. Учитывая эти свойства Compact GA, можно предложить его аппаратную реализацию в виде автомата, который синтезируется в большой программируемой интегральной схеме (ПЛИС, FPGA) и описан на языке VHDL или Verilog HDL. Следует заметить, что некоторым вопросам аппаратной реализации Compact GA посвящена работа [4]. В работе [5] предложена реализация алгоритма вероятностного алгоритма UMDA.

Целью статьи является следующее:

– синтез аппаратной последовательно-параллельной реализации генетического поиска, оптимально использующей логические ресурсы FPGA;

– построение реализации Compact GA с учетом возможности селекции решений (хромосом).

Параллельная аппаратная реализация. Вначале проанализируем полную параллельную реализацию алгоритма Compact GA.

Рассматривается n-битная проблема, которая определена как минимизация функции $F(X)$, где X – n-битный двоичный вектор. Входным параметром также является размер популяции m.

Генетический алгоритм Compact GA может быть описан следующим образом.

0. Для $i \in [1,n]$: $P_i = 0,5$.

1. Сгенерировать два двоичных вектора X и Y, для которых $p(\{X_i = 1\}) = P_i$ и $p(\{Y_i = 1\}) = P_i$.

2. Вычислить $F_X = F(X)$ и $F_Y = F(Y)$.

3.1. Если $F_X < F_Y$, то для $i \in [1,n]$, и всех таких $X_i \neq Y_i$ выполнить $P_i = P_i + X_i/m - (1 - X_i)/m$.

3.2. Иначе – для $i \in [1,n]$, и всех таких $X_i \neq Y_i$ выполнить $P_i = P_i + Y_i/m - (1 - Y_i)/m$.

4. Если существует такой P_i, что $0 < P_i < 1$, то – продолжить процесс (перейти к п.1), иначе – решение получено.

Из данного алгоритма видно, что основным элементом аппаратной реализации генетического алгоритма является генератор случайных чисел, который генерирует две последовательности из

двоичных чисел, с заданной вероятностью получения единичного значения $p(1) = p$ на выходе. Каждая выходная последовательность образует соответствующий бит вектора решения X и Y.

Структурная схема генератора случайных чисел показана на рисунке 1. В схеме LSFR – регистр сдвига разрядностью k с линейной обратной связью, реализующий генератор псевдослучайных чисел разрядностью k; RG – регистр разрядностью $k+1$ ($k = log_2 m$), который хранит вероятность генерации p; элемент помеченный как «=» – схема сравнения, генерирующая логическую 1, если выполняется $LSFR < RG$.

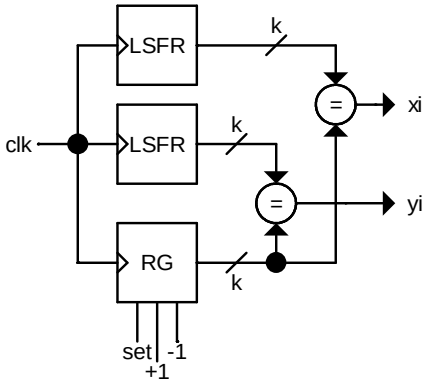

Рисунок 1 – Функциональная схема генератора одного бита векторов решения

Сложность реализации этого узла $S_R(k)$, выражаемая в логических ячейках FPGA, линейно зависит от k, примерно как $S_R(k) = 5k + 4$.

Поскольку вычисление целевой функции – отдельная задача, которая здесь не рассматривается, будем считать, что некоторая схема со сложностью S_F^* вычисляет за t_F^* тактов по векторам X и Y признак w. В случае, если $F(X) < F(Y)$, то $w = 1$, иначе – $w = 0$.

Согласно алгоритму Compact GA, после каждого оценивания решений X и Y осуществляется модификация вектора вероятностей P. В терминах аппаратной реализации, это означает, что входы +1 (обозначим этот сигнал как I_i) и –1 (D_i) зависят от сигналов X_i, Y_i и результата оценивания решений w. Соотношения, связывающие эти сигналы имеют вид:

$$\begin{cases} I_i = (X_i \oplus Y_i) \wedge (X_i w \vee Y_i \overline{w}), \\ D_i = (X_i \oplus Y_i) \wedge (\overline{X}_i w \vee \overline{Y}_i \overline{w}). \end{cases} \quad (1)$$

Поскольку каждая функция в (1) реализуется на одной логической ячейке FPGA, сложность узла составит – $S_C(k) = 2$.

Общая сложность $S(n,k)$ схемы зависит от параметров n и k следующим образом:

$$S(n,k) = n(S_R(k) + S_C(k)) + S_F^*(n,k) = 5nk + 6n + S_F^*(n,k).$$

Рассмотренная реализация оказывается самой быстродействующей из всех возможных. Если положить, что вектора решений X и Y оцениваются за один такт и модуль вычисления целевой функции представляет собой полностью комбинационную схему, то генерация одной популяции осуществляется за один такт синхросигнала, поступающего на тактовые входы регистров LSFR и RG.

Последовательно-параллельная аппаратная реализация. Параллельная реализация Compact GA оказывается очень эффективной. Однако не все преимущества архитектуры FPGA оказываются задействованы в данном случае (например, встроенная память). Также, существуют такие задачи, где высокое быстродействие параллельного алгоритма Compact GA не требуется, и необходимо найти некоторый компромисс между быстродействием и стоимостью (потреблением) FPGA.

В этих случаях эффективной является последовательно-параллельная реализация алгоритма Compact GA, описанная ниже.

Суть реализации заключается в том, что n битов вектора решения разбиваются на r блоков по $l = \lceil n/r \rceil$ бит в каждом. Каждый бит решения хранится в адресуемой однобитной ячейке, в которую заносится результат сравнения значения LSFR-регистра генератора псевдослучайных чисел с вероятностью P_i. Для оптимального использования ресурсов FPGA, вероятности P_i хранятся во встроенной RAM-памяти. Поскольку каждый блок работает независимо от других, то число r ограничивается количеством блоков встроенной в FPGA RAM-памяти.

Структура блока для формирования l битов решения показана на рисунке 2.

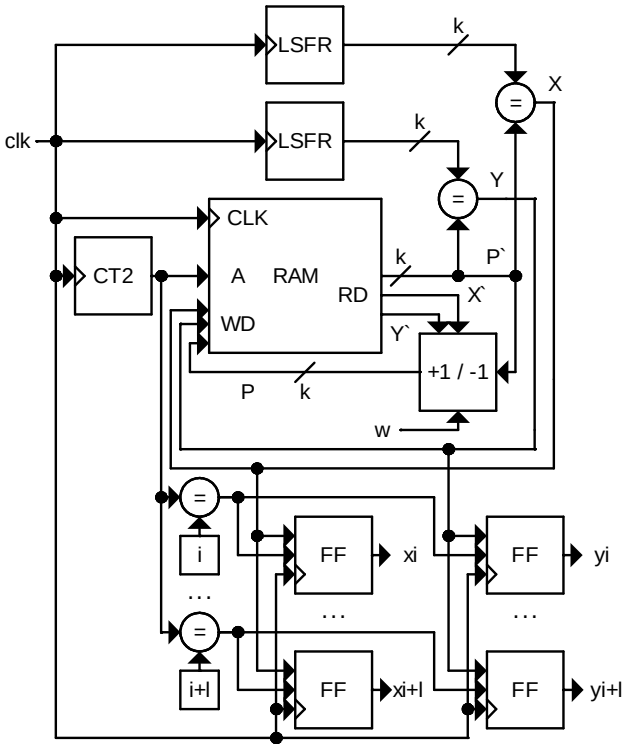

Рисунок 2 – Функциональная схема формирователя фрагмента решения

Основа модуля – ОЗУ (RAM). По адресу i в RAM-памяти хранится значение вероятности P_i вместе с битами векторов решений X_i и Y_i.

Рассмотрим процесс функционирования модуля. Диаграммы работы показаны на рисунке 3. Пусть два вектора решения X и Y оценены, и признак w условия $F(X) < F(Y)$ сгенерирован. Счетчик CT2, инкрементирующийся по спаду сигнала clk, устанавливает адрес i текущей ячейки памяти (на рисунке 2 – момент времени 1). Одновременно с этим, генераторы LSFR вырабатывают очередное случайное число.

После установления адреса i на входе A, по фронту clk на выходе памяти RD устанавливается вероятность P`= $P_i$ и значения X`= X_i и Y`= $Y_i$ (момент времени 2). Комбинационные схемы сравнения (обозначенные на рисунке 2 символом «=»), вырабатывают значения X и Y (момент времени 3). В этот же момент, логическая схема, обозначенная на рисунке 2 как «+1/–1», анализируя значения сигналов X`, Y` и признака w, модифицирует (уменьшает или увеличивает на 1) считанную из памяти вероятность P`, формируя новое значение P.

По спаду сигнала clk сформированные на линиях X и Y значения записываются в выбранную пару триггеров FF, хранящих биты векторов результата для оценивания (момент времени 4). В этот же момент в RAM-память по адресу i записываются значения P, X и Y.

Счетчик адреса CT2 увеличивает свое значение, и цикл повторяется для следующего бита решения.

Видно, что формирование всех l битов фрагмента решения осуществляется за $\tau = l$ тактов синхросигнала clk.

Оценим сложность всей реализации. ОЗУ использует встроенные в FPGA блоки памяти и поэтому не занимает

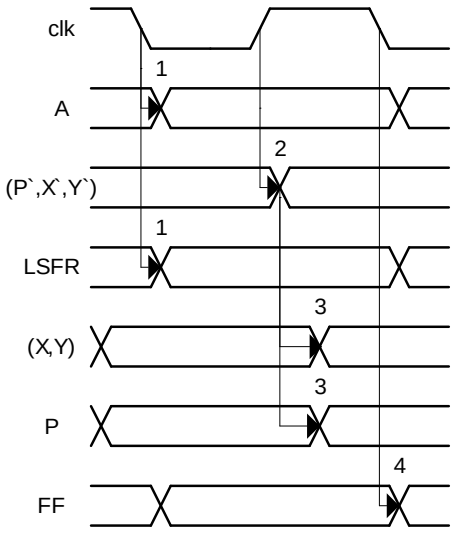

Рисунок 3 – Диаграммы работы формирователя фрагмента решения

логических ресурсов. Обозначим $q = \lceil log_2 l \rceil$. Счетчик CT2 всегда занимает q ячеек. Схемы сравнения, LSFR и инкрементор-декрементор в сумме занимают $5k + 4$ ячеек (как генератор одного бита в полностью параллельной реализации).

Рассмотрим логические затраты на реализацию линейки элементов памяти из триггеров FF и детекторов адреса. Каждый детектор адреса представляет собой схему, реализующую булеву функцию от q переменных, выход которой подключен к триггерному элементу. Можно показать, что сложность $S_{bit}(q)$ реализации одного бита линейки определяется как число логических элементов, с числом входов 4, минимально необходимых для реализации функции от q переменных: $S_{bit}(q) = \lceil \frac{q-1}{3} \rceil$.

Сложность $S_B(k,l)$ одного блока формирователя фрагмента решения определяется как сумма:

$$S_B(k,l) = 5k + 4 + 2l\lceil (\lceil log_2 l \rceil - 1)/3 \rceil + \lceil log_2 l \rceil .$$

Сложность $S(n,k,r)$ реализации схемы в целом оценивается как:

$$S(n,k,r) = 5kr + 4r + 2n\lceil (\lceil log_2 (n/r) \rceil - 1)/3 \rceil + r\lceil log_2 (n/r) \rceil + S_F^*(n,k) .$$

Об оптимальности последовательно-параллельной реализации. Будем рассматривать такие задачи, в которых $n = 2^a$, где $a \in \{1, 2...\}$. Очевидно, что любая другая задача может быть сведена к рассматриваемой. Для наиболее оптимального распределения ресурсов необходимо, чтобы все ячейки блоков памяти были заполнены, т.е. $l = 2^b$, где $b \in \{1, 2...\}$.

Обозначим l как: $l = n/r = 2^{log_2 n - log_2 r} = 2^{\gamma}$. Очевидно, что $\gamma \in Z$.

В таком случае, можно переписать выражение для сложности $S(n,k,\gamma)$:

$$S(n,k,\gamma) = 5kn/2^{\gamma} + 2n\lceil (\gamma - 1)/3 \rceil + \gamma n/2^{\gamma} + 4 + S_F^*(n,k) .$$

Рассматривая функцию $S(n,k,\gamma)$ как непрерывную от своих аргументов, можно убедиться в ее унимодальности на промежутке $0 < \gamma < log_2 n$. К сожалению, аналитически минимум $S(n,k,\gamma)$ по γ найти невозможно, и точное решение можно найти только численно. Следующее соотношение дает относительно точную оценку минимума $S(n,k,\gamma)$ по γ:

$$\gamma`_{opt} = \left\lceil \frac{1}{ln(2)} ln\left[\frac{3}{2}((5k+5) \cdot ln(2) - 1) \right] \right\rceil = \gamma(k) . \qquad (2)$$

Поскольку $\tau = l$, то $\tau = \lceil n/r \rceil$. Рассматривая оптимальность по времени решения, минимум τ достигается при разбиении результата на r_{opt} блоков, т.е.

$$r_{opt} = min\{r_{FPGA}, n\} ,$$

где r_{FPGA} – число блоков памяти в FPGA.

Рассмотрим два случая синтеза аппаратной реализации Compact GA для решения n-битной проблемы.

Для первой задачи задано ограничение на стоимость (сложность) через S и r в виде:

$$\begin{cases} S < S_{FPGA}, \\ r < r_{FPGA}, \end{cases}$$

где S_{FPGA} – максимальная сложность схемы, r_{FPGA} – максимальное число блоков памяти. Требуется найти такую структуру, которая бы удовлетворяла этим ограничениями и обладала максимальной производительностью ($\tau \to \min$). Если $S_{FPGA} \geq 5nk + 6n + S_F^*(n,k)$, то эта задача решается с помощью полностью параллельной реализации Compact GA. В противном случае, используется последовательно-параллельная реализация с разбиением результата на r_{FPGA} блоков.

В другом случае, задано ограничение на минимальное быстродействие τ:

$$\tau < \tau_{\max} .$$

Требуется найти такую структуру, которая бы удовлетворяла этому ограничению и обладала минимальной сложностью: $S \to \min$. Поскольку $\tau = l = n/r$, то $\gamma_{\max} = n \cdot 2^{-n/\tau_{\max}}$. Вычислив по формуле (2) значение $\gamma`_{opt}$, получим два варианта: $\gamma_{\max} \geq \gamma`_{opt}$ и $\gamma_{\max} < \gamma`_{opt}$. Если $\gamma_{\max} \geq \gamma`_{opt}$, то реализуем схему с $\gamma = \gamma`_{opt}$. В противном случае для реализации берем $\gamma = \gamma_{\max}$.

Применение селективной техники. В реализации, рассмотренной выше, использовались два вектора решения X и Y. Как показано в [3], можно уменьшить число вычислений целевой функции, введя селекцию из v векторов решений $X_1,... X_v$.

Для эффективной аппаратной реализации можно переформулировать соответствующий алгоритм, приведенный в [3], следующим образом.

0. Для $i \in [1,n]$: $P_i = 0,5$.

1. Сгенерировать v двоичных векторов $X = \{X^1,...X^v\}$, для каждого из которых $p(\{X_i^k = 1\}) = P_i$.

2. Для $h \in [1, C_v^2]$ выполнять

2.1. По значению h сформировать индексы $i = i(h)$ и $j = j(h)$ пары решений F_i и F_j.

2.2. Если $F_i < F_j$, то $w_h = 1$, иначе $w_h = 0$.

3. Для $k \in [1,n]$, вычислить $\Delta P_k = \Delta(w, X_k^i)$ и скорректировать $P_k = P_k + \Delta P_k$.

4. Если существует такой P_k, что $0 < P_k < 1$, то – продолжить процесс, иначе – решение получено.

Рассмотрим общую структуру системы, показанную на рисунке 4.

Как и для случая $v = 2$, биты результата хранятся в блоках памяти (для коррекции вероятностей) и в линейке триггеров-защелок (для оценки решения).

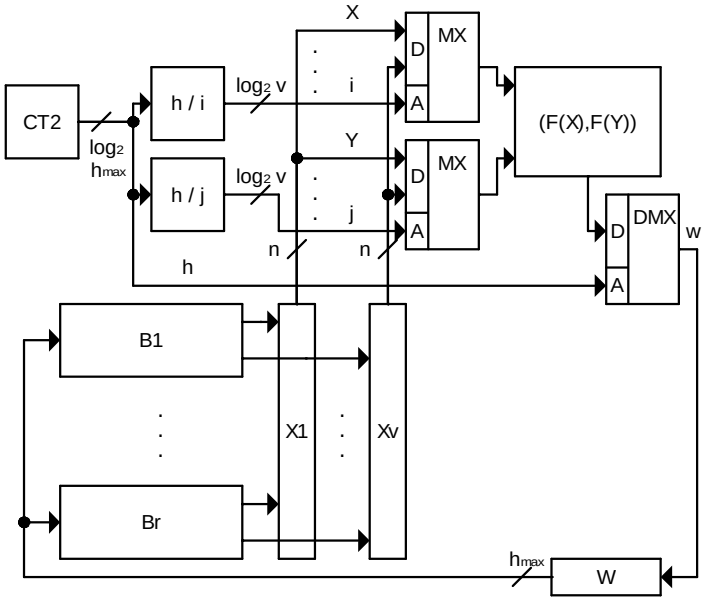

Рисунок 4 – Структура реализации с селекцией из множества векторов решений

На рисунке 4 блоки X1-Xv – векторы решений. Компоненты X1-Xv генерируются с помощью элементов B1-Br, каждый из которых содержит ОЗУ и v генераторов случайных чисел. Каждый Bi осуществляет генерацию $l = \lceil n/r \rceil$ одноименных бит в X1-Xv.

Как только векторы результата X1-Xv сгенерированы, начинает работу счетчик CT2, который на своем выходе последовательно выдает коды h пар результата от 0 до $h_{max} = C_v^2 - 1$. Преобразователи h/i и h/j формируют по коду h индексы i и j для выбора двух векторов решений. Через мультиплексоры MX значения векторов решений подаются на схему оценивания (F(X),F(Y)) в качестве аргументов X и Y. Схема оценивания вырабатывает признак $w = 1$ при $F(X) < F(Y)$. Далее значение w передается через демультиплексор DMX на регистр W. Таким образом, после перебора и оценивания всех пар, оказывается сформирован вектор W, в котором каждый бит w_i представляет результат оценки i-ой пары.

Преобразователи h/i и h/j задают табличное отображение кодов h пар в индексы i и j ($h = 0$, $(i, j) = (0,1)$; $h = 1$, $(i, j) = (0,2)$; и т.д.).

Блок генерации фрагмента решения Bi подобен структуре, изображенной на рисунке 3 (для $v = 2$) с тем отличием, что содержит $v > 2$ генераторов LSFR и схем сравнения для формирования компонентов каждого вектора решения X^i.

Модуль коррекции вероятности P_k на $\Delta(w, X_k^i)$, заменяющий управляемый инкрементор-декрментор в реализации на рисунке 2, состоит из системы элементов, представляющих собой модули для вычисления значений сигналов I_i и D_i по выражению (1), выходы которых присоединены к управляемым инкременторам-декременторам. Всего в схеме содержится h_{max} таких элементов, каждый из которых анализирует значение w_h и обрабатывает соответствующую w_h пару $(x_i, x_j) = (X_k^i, X_k^j)$ векторов решения. Например, для $v = 4$ схема коррекции P_k осуществляет следующее преобразование:

$$P_k = P`_k + I(x_0, x_1, w_1) + I(x_0, x_2, w_2) + I(x_0, x_3, w_3) + I(x_1, x_2, w_4) + I(x_1, x_3, w_5) + I(x_2, x_3, w_6) -$$
$$- D(x_0, x_1, w_1) - D(x_0, x_2, w_2) - D(x_0, x_3, w_3) - D(x_1, x_2, w_4) - D(x_1, x_3, w_5) - D(x_2, x_3, w_6),$$

где $I(x_i, x_j, w_h) = (x_i \oplus x_j) \wedge (x_i w_h \vee x_j \overline{w_h})$ и $D(x_i, x_j, w_h) = (x_i \oplus x_j) \wedge (\overline{x_i} w_h \vee \overline{x_j}\,\overline{w_h})$ согласно (1); P_k – значение вероятности на предыдущем шаге.

Оценим сложность рассмотренной реализации.

Сложность одного блока формирования фрагмента решения $S_B(v, k, l)$ может быть определена следующим образом:

$$S_B(v, k, l) = v(2k + 2) + vl\lceil(\lceil log_2 l\rceil - 1)/3\rceil + \lceil log_2 l\rceil + 6C_v^2.$$

Сложность всей схемы $S(v, n, k, r)$ находится как:

$$S(v, n, k, r) = vr(2k + 2) + vn\lceil(\lceil log_2(n/r)\rceil - 1)/3\rceil + r\lceil log_2(n/r)\rceil + 6rC_v^2 + S_F^*(n, k) + vn.$$

Некоторые практические результаты. Предложенные структуры для аппаратной реализации генетического поиска были опробованы на ПЛИС (FPGA) EP2C20F484C7N семейства Cyclone II. Все схемы описаны на языке Verilog HDL и в качестве логического компилятора был использован Quartus II 6.0.

Приведем результаты простой полностью параллельной реализации генетического алгоритма. Тактовая частота устройства f = 100 МГц. Размер популяции $m = 64$. Рассматривается квадратичная n-битная проблема: $F(X) = \sum_{i=1}^{N} x_i^2$, где для представления x_i отводится 10 битов и $N = n/10$.

Приведем результаты аппаратной реализации и быстродействия системы для различных структур и разного числа n (таблица 1).

Таблица 1 – Результаты различных реализаций алгоритма Compact GA

n	S_F^*, лог. эл.	$S_{\|\|}$, лог. эл.	$\tau_{\|\|}$, мкс	S_S, лог. эл.	τ_S, мкс	S_τ, лог. эл.	τ_τ, мкс
30	108	1099	10	261	320	1645	10
60	242	2242	13	484	416	1872	26
100	426	3616	18,2	892	582,4	2029	36,4
200	871	7273	23,6	1707	755,2	2751	94,4
400	1758	14428	30	3400	960	3972	300

В таблице 1 использованы следующие обозначения: n – число битов вектора решения; S_F^* – сложность схемы оценивания решений в логических элементах FPGA; $S_{\|}$ – сложность полностью параллельной реализации Compact GA; $\tau_{\|}$ – время минимизации функции с применением параллельной реализации; S_S – сложность реализации, полученной при оптимальном по сложности разбиении результата, на блоки по $2^{\gamma`_{opt}} = 32$ бита; τ_S – время минимизации функции при оптимальном по сложности разбиении результата; S_τ – сложность реализации, полученной при разбиении результата на $r = r_{FPGA} = 50$ блоков для максимизации быстродействия; τ_S – время минимизации функции при разбиении результата на $r = 50$ блоков.

Как видно из результатов, даже для 400-битной проблемы время поиска минимума не превышает 1 мс, что позволяет применять рассмотренные аппаратные реализации в системах оптимизации и управления реального времени. Использование последовательно-параллельной реализации позволяет значительно снизить аппаратные затраты для приложений, где не требуется сверхвысокое быстродействие.

Выводы. Предложены и проанализированы различные варианты аппаратной реализации генетического алгоритма Compact GA для оптимизации функций. Рассмотрена полностью параллельная реализация, последовательно-параллельный подход и применение селекции. Предложенные соотношения для оценки быстродействия и сложности позволяют выбрать наиболее оптимальный вариант реализации для заданной задачи.

Дальнейшие исследования будут направлены на применение конвейеризации к процессам генерации и оценивания решений, частичному (сокращенному) оцениванию результатов. Предложенный подход с небольшими изменениями может быть применен для реализации некоторых других вариантов генетических алгоритмов (в первую очередь, для SG-Clans алгоритма).

Библиографический список

1. Martinez M. Generalized predictive control using genetic algorithms (GAGPC) / M. Martinez, J.S. Senent, X. Blasco // Proceedings of Engineering Applications of Artificial Intelligence, Castellón, Spain, June 1 – 4, 1998. — Castellón, 1998. — V. 11. — P. 355 – 367.

2. Rudnick E.M. Application of Simple Genetic Algorithm to Sequential Circuit Test Generation / E.M. Rudnick, J.G. Holm, D.G. Saab, J.H. Patel // Proc. European Design & Test Conf, Paris, France, September 17 – 20, 1994. — Paris, 1994. — P. 40 – 45.

3. Harik G. The Compact genetic Algorithm / G. Harik, F. Lobo and D. Goldberg // IEEE Transactions on Evolutionary Computation. — 1999. — V. 3. — P. 287 – 309.

4. Chatchawit A. A Hardware Implementation of the Compact Genetic Algorithm / A. Chatchawit and C. Prabhas // Proceedings of the 2001 IEEE Congress on Evolutionary Computation, Seoul, Korea, May 27 – 30, 2001. — Seoul, 2001. — P. 624 – 629.

5. Гудилов В.В. Методы повышения эффективности вероятностных алгоритмов, адаптированные к возможности динамических изменений параметров функционирования на примере аппаратной реализации генетического алгоритма UMDA / В.В. Гудилов, В.М. Курейчик // Перспективные информационные технологии и интеллектуальные системы: сб. научн. тр. — Таганрог, 2005. — № 1 (21). — С. 42 – 54.

Поступила в редакцию 24.10.2007 г.