УДК 681.3

ЭВРИСТИЧЕСКИЙ ГЕНЕТИЧЕСКИЙ АЛГОРИТМ ОПТИМИЗАЦИИ ВЫЧИСЛИТЕЛЬНЫХ КОНВЕЙЕРОВ

А.А. ПРИХОЖИЙ[1], А.М. ЖДАНОВСКИЙ[1], О.Н. КАРАСИК[1], М. МАТТАВЕЛЛИ[2]

[1]*Белорусский национальный технический университет, Республика Беларусь*

[2]*École polytechnique fédérale de Lausanne, Швейцария*

Поступила в редакцию 14 ноября 2016

Аннотация. Проведен анализ вычислительных конвейеров и методов их оптимизации. Рассмотрен класс потоковых конвейеров, не использующих разделение вычислительных ресурсов и достигающих наибольшей производительности. Для этого класса рассмотрены задачи оптимизации конвейеров, решаемые для случайной логики в процессе высокоуровневого синтеза по поведенческим спецификациям на языках программирования и описания аппаратуры. Исследован эвристический генетический алгоритм оптимизации, применимый к проектам большого размера.

Ключевые слова: поток данных, вычислительный конвейер, высокоуровневый синтез, оптимизация.

Abstract. An analysis of computational pipelines and their optimization methods has been performed. A class of dataflow pipelines that do not use resource sharing and obtain high throughput has been considered. Pipeline optimization tasks being solved during high-level synthesis from random logic behavioral specifications in programming and hardware description languages have been considered. A heuristic genetic optimization algorithm which is capable of handling large designs has been proposed and investigated.

Keywords: data flow, computational pipeline, high-level synthesis, optimization.

Doklady BGUIR. 2017, Vol. 103, No. 1, pp. 34-41
Heuristic genetic algorithm of computational pipelines optimization
A.A. Prihozhy, A.M. Zhdanouski, O.N. Karasik, M. Mattavelli

Введение

Конвейер является распространенной архитектурой, повышающей производительность вычислений во многих прикладных областях. Конвейеризация – это процесс преобразования поведенческой спецификации системы, в результате которого операторы распределяются по ступеням конвейера, обрабатывающим движущийся через них поток данных. Ранние методы построения и оптимизации конвейеров и многие недавно разработанные алгоритмы [1, 2] нацелены на эффективное разделение вычислительных ресурсов между операторами. Это сериализует вычисления и снижает производительность системы, поэтому с развитием технологий интегральных микросхем появились конвейерные архитектуры, не использующие разделение ресурсов [3]. Для таких архитектур важным является предварительное преобразование спецификаций «поток управления / поток данных» к спецификациям потока данных [4, 5]. При оптимизации конвейеров по потоковым спецификациям решаются две главные задачи: выбор компонентов для реализации операторов и размещение конвейерных буферов. Первая задача получила решение в [3]. Для решения второй задачи предложен точный оптимальный алгоритм в [6, 7], однако в силу комбинаторного характера он применим для конвейеров небольшого размера (до 100 операторов и 5 ступеней конвейера). В настоящей

статье предлагается эвристический генетический алгоритм, способный оптимизировать многоступенчатые конвейеры для спецификаций, состоящих из тысяч операторов.

Методика эксперимента

Для синтеза и оптимизации конвейера поведенческая спецификация преобразуется к единому линейному базовому блоку путем расщепления выражений, раскрутки циклов, расщепления условных операторов [4]. Базовый блок транслируется в граф потока данных, по которому строится ряд отношений: зависимостей переменных и операторов, информационных зависимостей между операторами, взаимной исключаемости операторов, предшествования операторов, принадлежности операторов разным ступеням конвейера (или конфликтов операторов на ступенях конвейера). Бинарным отношениям ставятся в соответствие графовые модели. Отношение конфликтов строится по времени работы ступени конвейера путем оценки временных задержек исполнения операторов и вычисления критических путей на графе предшествования операторов [6].

Конвейер характеризуется рядом параметров: числом ступеней, временем работы одной ступени, суммарным размером всех конвейерных буферов *BS* (buffer size) и др. Отображение операторов на ступени, называемое также конвейерным планом, описывается функцией *stage*. Количество возможных функций растет экспоненциально в зависимости от размера спецификации, числа ступеней конвейера и мобильности операторов на ступенях. Каждый конвейерный план обладает своим значением *BS*. План ASAP назначает операторы на допустимые ранние ступени, план ALAP назначает операторы на допустимые поздние ступени [3]. Оптимальный план [6], генерируемый алгоритмом LCSBB, выбирает наиболее подходящую ступень для каждого оператора и дает минимальное значение *BS*.

Размер буферов оценим нижней границей *bslb* (*buffer size lower bound*), которая для частично построенного конвейерного плана оценивается уравнением

$$bslb(stage) = \sum_{v \in V} size(v) \times lifetime(v), \qquad (1)$$

где V – множество переменных, являющихся входами и выходами операторов; $size(v)$ – размер переменной v; $lifetime(v)$ – время жизни переменной v, т.е. диапазон ступеней конвейера, на котором значение переменной вычисляется и используется. Нижняя граница времени жизни в (1) определяется уравнением $lifetime(v) = late(v) – early(v)$, где $late(v)$ – нижняя граница наиболее поздней ступени конвейера для v; $early$ (v) – верхняя граница наиболее ранней ступени. Значение $late(v)$ оценивается как максимум $early(q)$ на ступенях наиболее раннего выполнения операторов-потребителей $q \in cons(v)$ переменной v. Значение $early(v)$ оценивается как минимум $late(q)$ на ступенях наиболее позднего выполнения операторов-производителей $q \in prod(v)$ переменной v.

Если оператор q уже назначен на ступень, то $early(q) = late(q) = stage(q)$, в противном случае $early(q)$ и $late(q)$ рассчитываются для каждого частично построенного конвейерного плана через $stage(p)$ уже назначенных на ступени операторов и через $asap(p)$ и $alap(p)$ еще не назначенных операторов.

Величина *bslb* равна точному значению размера буферов для полностью построенного плана и используется как эвристика при выборе ступени конвейера, на которую назначается очередной оператор в частично построенном плане. Второй эвристикой является мобильность $mob(p)$ оператора p на множестве ступеней.

Вначале $mob(p) = alap(p) – asap(p)+1$, на последующих шагах $mob(p) = late(p) – early(p)+1$. Третьей эвристикой является абсолютная разность суммарного размера входов $sizeIn(p)$ и суммарного размера выходов $sizeOut(p)$ оператора p: $dios(p) = |sizeIn(p) – sizeOut(p)|$. Эвристический динамический алгоритм HDA (heuristic dynamic algorithm) минимизации *BS* представлен на рис. 1, где N – множество операторов, $bslb(p,s,P)$ – *bslb* для операторов из P, уже назначенных на ступени конвейера, и оператора p, назначаемого на ступень s. Он включает цикл, на каждой итерации которого выбирает один не назначенный оператор $p \in Q$ и выбирает ступень s, на которую p назначается. Выбор лучшего оператора p и ступени s выполняется посредством веса χ_p:

$$\chi(p) = \sum_{i=1}^{\kappa} \omega_i \cdot \rho_i(p). \qquad (2)$$

Здесь $\rho_i(p)$ – эвристический параметр; ω_i – эвристический коэффициент; k – число эвристических параметров. Для эвристических коэффициентов должно выполняться равенство: $\sum_{i=1\ldots k} \omega_i = 1$. Значение эвристического параметра $\rho_i(p)$ должно варьироваться между 0 и 1, а с большим значением параметра должны ассоциироваться лучшие характеристики конвейера. Оператор $p \in Q$ с максимальным значением $\chi(p)$ выбирается в качестве следующего кандидата на планирование.

Инициализировать состояние оптимизации.
Присвоить $P := \varnothing$ и $Q = N$.
Пока $Q \neq \varnothing$ выполнить {
  Для каждого оператора $q \in Q$ { Вычислить динамические параметры $\rho(p)$ и вес $\chi(q)$. }
  Выбрать оператор p с максимальным весом $\chi(p)$ из Q.
  Для каждой ступени конвейера $t \in \{early(p) \ldots late(p)\}$ { Оценить $bslb(p,t,P)$. }
  Выбрать ступень s с минимальным размером буферов $bslb(p,s,P)$ для оператора p и
    присвоить $stage(p) := s$.
  Присвоить $P := P \cup \{p\}$ и $Q := Q / \{p\}$.
  Обновить $early(q)$ и $late(q)$ для всех $q \in Q$, а также $early(v)$ и $late(v)$ для всех $v \in V$.
}
Вернуть результат *stage*.

Рис.1. Эвристический динамический алгоритм HDA оптимизации конвейера

Для минимизации размера буферов используются четыре параметра $\rho_1(p)$–$\rho_4(p)$. Первым $\rho_1(p)$ является дополнение до 1 относительной мобильности оператора p на множестве ступеней. Он оценивается через абсолютную мобильность $mob(p)$, минимальную и максимальную мобильность среди всех операторов из Q. Низкое значение мобильности p влечет высокое значение $\rho_1(p)$, которое заставляет назначать на ступени в первую очередь трудно планируемые операторы. Вторым параметром $\rho_2(p)$ является относительная максимальная разность *dbslb* значений *bslb* для оператора p на множестве доступных ступеней конвейера. Она оценивается через разность минимального и максимального значений *bslb* на ступенях оператора при взятии максимального значения на множестве всех операторов. Большее значение разности *dbslb* влечет большее значение $\rho_2(p)$, которое отдает предпочтение оператору, планирование которого минимально увеличит значение размера буферов. Третьим параметром $\rho_3(p)$ является дополнение до 1 относительного увеличения *mbslb* минимального по ступеням *bslb* оператора p среди всех операторов из Q. Он показывает минимальное увеличение *bslb* после назначения выбираемого оператора на лучшую ступень в сравнении с худшим оператором из Q. Четвертым параметром $\rho_4(p)$ является относительная разность *dios* между размером входных операндов и размером выходных операндов оператора p. Он показывает значимость перемещения операторов по ступеням: если размер выходов больше размера входов, оператор должен перемещаться к ранней ступени, иначе к поздней ступени.

Значимость параметра $\rho_i(p)$, $i = 1 \ldots k$ в весе χ_p определяется коэффициентом ω_i: чем выше значение, тем более значим параметр. Вычисление наилучших значений коэффициентов является сложной оптимизационной задачей со многими локальными оптимумами, поскольку вариация значений коэффициентов приводит к изменению структуры частичного плана и, соответственно, к изменению эвристических параметров конвейера. Для решения задачи предлагается следующий генетический алгоритм GA. Хромосому представим вектором эвристических коэффициентов $\omega = (\omega_1, \ldots, \omega_k)$, в котором ω_i есть ген. Популяция есть множество хромосом, существующих во время работы GA, а поколение есть множество хромосом, существующих на одной итерации GA. Фитнес функция $F(\omega)$ хромосомы ω определяется как разность между наибольшим размером буферов $BS(\omega^{worst})$ на множестве всех хромосом и размером буферов $BS(\omega)$ хромосомы ω, вычисляемым алгоритмом HDA. Хромосомы популяции сортируются по убыванию функции $F(\omega)$.

Для оптимизации эвристических коэффициентов предлагаются следующие генетические операции. Они построены с учетом существующих подходов, но отражают особенности решаемой задачи. Операция селекции, используемая при выборе хромосом для скрещивания и мутации, реализуется по правилу рулетки (FPS – fitness proportionate selection), выбирающему с большей вероятностью те хромосомы, которые имеют большее значение фитнес функции. Операция селекции, используемая для генерации нового поколения, заменяет лучшим потомком худшего родителя (WPS – worst parent selection) или худшую хромосому поколения (WIS – worst individual selection).

Для выполнения операции скрещивания исследованы два кроссовера: 1) одноточечный однородный HUX (half uniform crossover), порождающий два потомка; 2) взвешенно усредняющий SOX (single offspring crossover), порождающий одного потомка. Кроссовер HUX принимает две родительские хромосомы ω^1 и ω^2 и равновероятно разбивает множество $K = \{1, \ldots, k\}$ индексов генов на два равных подмножества K^1 и K^2. Он является частично подходящим и не дает корректных потомков рекомбинацией генов, требуя их нормализацию, поскольку сумма коэффициентов может отличаться от 1. Для этого кроссовера различаются два варианта.

Вариант 1. Значения фитнес функции для ω^1 и ω^2 приблизительно равны, $F(\omega^1) \approx F(\omega^2)$. HUX стремится сохранить генофонд ω^1 в первом потомке ω^3, а генофонд ω^2 во втором потомке ω^4. Потомок ω^3 конструируется из генов хромосомы ω^1 с индексами из K^1, и нормализованных генов хромосомы ω^2 с индексами из K^2. Потомок ω^4 конструируется из генов хромосомы ω^2 с индексами из K^2, и нормализованных генов хромосомы ω^1 с индексами из K^1. Для нормализации генов используются следующие величины: $a = \sum_{i \in K^1} \omega^1_i$, $b = \sum_{i \in K^1} \omega^2_i$, $\gamma^1 = (1-a)/(1-b)$, $\gamma^2 = b/a$. Величина γ^1 используется для нормализации генов первого потомка: $\omega^3{}_i = \gamma^1 \times \omega^2{}_i$ для $i \in K^2$. Величина γ^2 используется для нормализации генов второго потомка: $\omega^4{}_i = \gamma^2 \times \omega^1{}_i$ для $i \in K^1$.

Вариант 2. Значение фитнес функции $F(\omega^1)$ значительно превосходит значение фитнес функции $F(\omega^2)$. HUX стремится сохранить генотип ω^1 в обоих потомках. Первый потомок остается прежним, ω^3. Второй потомок ω^5 конструируется из генов хромосомы ω^1 с индексами из K^2 и нормализованных генов хромосомы ω^2 с индексами из K^1. Нормализация генов выполняется как $\omega^5{}_i = \omega^2{}_i / \gamma^2$ для $i \in K^1$.

Иллюстрация двух вариантов кроссовера HUX приведена на рис. 2. В варианте 1 значения 175 и 173 фитнес функции двух родителей близки, следовательно, потомки ω^3 и ω^4 стремятся сохранить генотипы обоих родителей. В варианте 2 значение 175 фитнес функции первого родителя значительно выше значения 92 фитнес функции второго родителя, и следовательно каждый потомок ω^3 и ω^5 стремится сохранить генотип первого родителя.

Кроссовер SOX принимает две родительские хромосомы ω^1 и ω^2 и производит одного потомка. Для этого он вычисляет веса α и β: $\alpha = F(\omega^1) / (F(\omega^1) + F(\omega^2))$, $\beta = 1 - \alpha$. Потомок ω вычисляется в виде вектора взвешенных сумм родительских генов: $\omega_i = \alpha \times \omega^1{}_i + \beta \times \omega^2$, для $i = 1 \ldots k$.

Сумма коэффициентов в полученной хромосоме равна 1. Ее гены ближе к генам первого родителя в случае $F(\omega^1) > F(\omega^2)$ и ближе к генам второго родителя в противном случае. Кроссовер SOX сканирует пространство, близлежащее к родителю с лучшим значением фитнес функции. Иллюстрация SOX дана на рис. 2. Фитнес функции родителей имеют значения 175 и 92 соответственно, отсюда $\alpha = 0{,}655$ и $\beta = 0{,}345$, а потомок ближе к первому родителю.

Мутация хромосомы ω^1 выполняется операцией TGM путем модификации двух генов (two gene mutation). Гены $\omega^1{}_i$ и $\omega^1{}_j$ выбираются случайным образом. Гены ω_i и ω_j потомка ω вычисляются с использованием коэффициента мутации ε, чье значение удовлетворяет неравенству $0 < \varepsilon < 1$: $\delta = \varepsilon \times \omega^1{}_i$, $\omega_i = \omega^1{}_i - \delta$, $\omega_j = \omega^1{}_j + \delta$. Мутация TGM иллюстрируется на рис. 2.

Значение случайно выбранного коэффициента $\omega_2 = 0{,}47$ уменьшилось на 0,14 при $\varepsilon = 0{,}3$, значение коэффициента ω_4 увеличилось на 0,14.

<table>
<tr><td colspan="2">HUX:
ω^1=(0.31, 0.47, 0.09, 0.13), ω^2=(0.25, 0.14, 0.53, 0.08)
K^1={1, 3}, $K\backslash K^1$={2, 4}
a=0.31+0.09=0.4, b=0.25+0.53=0.78
γ^1=(1−0.4)/(1−0.78)=2.73, γ^2=0.78/0.4=1.95
ω^3=(0.31, 0.38, 0.09, 0.22)</td><td rowspan="2">SOX:

ω^1=(0.31, 0.47, 0.09, 0.13)
ω^2=(0.25, 0.14, 0.53, 0.08)
$F(\omega^1)$=175
$F(\omega^2)$=92
α=175/(175+92)=0.655
β=1−0.655=0.345
ω=(0.29, 0.36, 0.24, 0.11)</td><td rowspan="2">TGM:

ε=0.3
ω=(0.31, 0.47, 0.09, 0.13)
i=2, j=4
δ=0.47*0.3=0.14
ω_2=0.47−0.14=0.33
ω_4=0.13+0.14=0.27
ω=(0.31, 0.33, 0.09, 0.27)</td></tr>
<tr><td>Вариант 1:
$F(\omega^1)$=175, $F(\omega^2)$=173
ω^4=(0.60, 0.14, 0.18, 0.08)</td><td>Вариант 2:
$F(\omega^1)$=175, $F(\omega^2)$=92
ω^5=(0.13, 0.47, 0.27, 0.13)</td></tr>
</table>

Рис. 2. Иллюстрация скрещивания HUX, скрещивания SOX и мутации TGM

На рис. 3 представлено цельное описание алгоритма GA минимизации размера конвейерных буферов. Алгоритм включает инициализацию и цикл, на каждой итерации которого формируется новое поколение конвейеров посредством генетических операций. Следующая генетическая операция выбирается посредством вероятностей p_{cross} для кроссовера и p_{mut} для мутации. Условием выхода из цикла является отсутствие улучшений в лучшей хромосоме для малых конвейеров и истечение заданного процессорного времени для больших конвейеров. GA используется в двух режимах: 1) реальная оптимизация конвейера и 2) накопление знаний о лучших значениях эвристических коэффициентов. Он применим для конвейеризации поведенческих спецификаций размером 5000 и более операторов.

1. Построить начальную популяцию. Для очередной хромосомы: сгенерировать k–1 случайных чисел μ_i, i=1…k–1 в диапазоне [0,1], упорядочить числа по возрастанию, вычислить хромосому в виде $\omega=(\mu_1, \mu_2-\mu_1,\ldots,\mu_{k-1}-\mu_{k-2}, 1-\mu_{k-1})$.
   Выполнить алгоритм HDA для каждой хромосомы ω, интерпретируемой как вектор эвристических коэффициентов, получить $BS(\omega)$, найти наихудшую хромосому, вычислить фитнес функцию $F(\omega)$ и упорядочить хромосомы по убыванию $F(\omega)$.
2. while (not Условие_выхода) do
   3. Случайно выбрать генетическую операцию.
   4. Случайно выбрать родителей посредством операции селекции FPS.
   5. Выполнить кроссовер HUCX или SOCX и получить два потомка.
   6. Выполнить операцию мутации TGM.
   6. Выполнить алгоритм HDA для каждого потомка ω и получить значение $F(\omega)$.
   7. Выполнить операцию селекции WPS или WIS и сгенерировать новое поколение.
end while
8. Результатом является лучшая хромосома из последнего поколения.

Рис. 3. Генетический алгоритм минимизации размера конвейерных буферов

Результаты и их обсуждение

Оценка качества работы эвристического генетического алгоритма выполнена на фильтре Байера, обратном дискретном косинусном преобразовании и других проектах. Он дает размер конвейерных буферов, отличающийся от глобального оптимума на 2 % в среднем. На больших проектах, таких как прямое дискретное косинусное преобразование, GA нельзя сравнить с точным алгоритмом LCSBB, поскольку последний не может получить решение за реальное процессорное время. Поэтому он сравнивается с алгоритмами ASAP и ALAP.

Ниже приведены результаты экспериментов, полученные на пяти крупных проектах TB1000-TB5000, состоящих из тысяч операторов. Коэффициентам $\omega_1,\ldots,\omega_4$ соответствуют оси четырехмерного пространства оптимизации. В первом эксперименте интервал [0, 1] на каждой оси разбит с шагом 0,05 на 20 отрезков. Все пространство представлено 1771 точками, являющимися допустимыми комбинациями значений коэффициентов. Рис. 4, представляющий зависимость $BS(\omega)$ от вектора ω, полученного алгоритмом HDA для 3-ступенчатого конвейера, доказывает, что размер буферов имеет много локальных минимумов, а задача оптимизации вектора ω является сложной вычислительной проблемой. Рис. 5 слева свидетельствует об эффективности эвристики *drslb*, т.к. среднее значение $BS(\omega)$ уменьшается с увеличением ω_2. Эффективность других эвристик уменьшается в порядке *dios*, *mob* и *mrslb*. Рис. 5 справа показывает, что фиксация значения ω_1 или ω_3 не локализует значение $BS(\omega)$, а подходящий выбор значений ω_2 или ω_4 сокращает разброс $BS(\omega)$ и ускоряет поиск минимального значения.

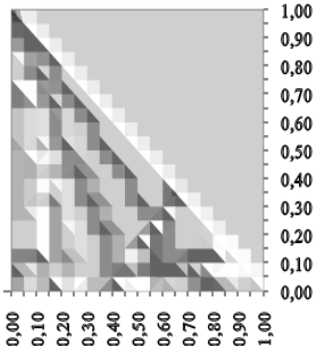

Рис. 4. Размер буферов в диапазоне от 871 до 1163 bit для 3-ступенчатого конвейера TB1000 в зависимости от ω_4 (*dios*) – горизонтальная ось и ω_3 (*mrslb*) – вертикальная ось при $\omega_1 = 0$ (*mob*) и $\omega_2 = 1-\omega_1-\omega_3-\omega_4$ (*drslb*)

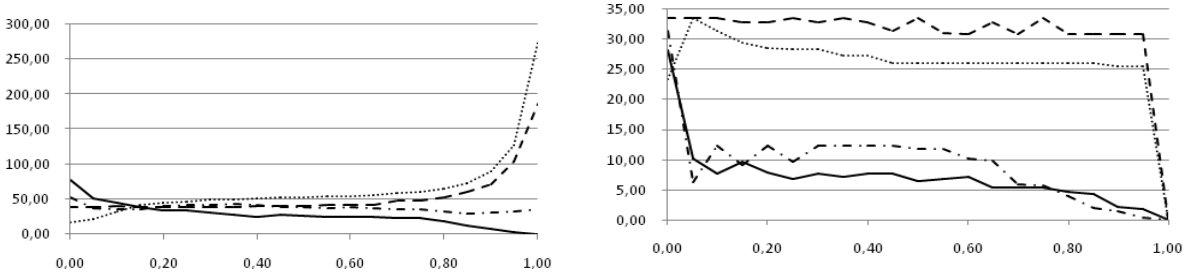

Рис. 5. Средний *BS*(ω) минус 888 в зависимости от коэффициента ω_1–*mob* (пунктирная), ω_2–*drslb* (сплошная), ω_3–*mrslb* (точечная), ω_4–*dios* (пунктирно-точечная) (слева) и разброс *BS*(ω) в процентах в зависимости от коэффициента ω_1, ω_2, ω_3, ω_4 (справа)

Во втором эксперименте многократное применение GA к конвейерам с различным числом ступеней позволило построить кумулятивные функции распределения вероятностей (CDF) лучших значений коэффициентов ω (рис. 6, слева). Средними значениями лучших коэффициентов являются $\omega_2 = 0{,}466$, $\omega_1 = 0{,}292$, $\omega_4 = 0{,}186$ и $\omega_3 = 0{,}056$. Они показывают важность соответствующих эвристик в критерии (2). Кроме того, CDF есть эффективное средство оптимизации конвейера и генерации начальной популяции в GA. Рис. 6 справа сравнивает генетические операции при увеличении размера популяции. Кроссовер SOX дает меньший размер буферов на первых хромосомах, а при увеличении размера популяции лучшим является HUX. Использование CDF повышает качество результатов оптимизации на начальных итерациях GA. Рис. 7 слева сравнивает GA с двумя известными алгоритмами минимизации буферов [3]: ASAP, известный как нисходящий обход графа потока данных, и ALAP, известный как восходящий обход графа потока данных. ASAP проигрывает GA от 27,32 % до 51,41 %, ALAP проигрывает от 51,71 % до 105,43 %. Рис. 7 справа показывает рост процессорного времени и его среднеквадратичного отклонения для алгоритма HDA-GA в зависимости от размера проекта на Intel® Core™ i3. GA способен оптимизировать многоступенчатые конвейеры за 2 min CPU для проектов, включающих более 5000 операторов.

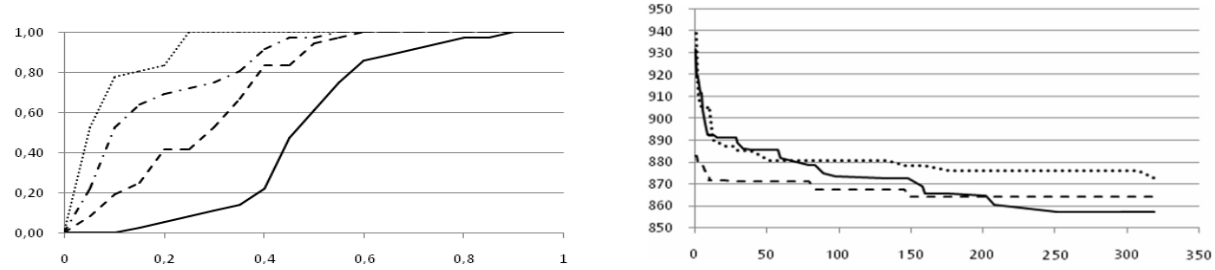

Рис. 6. Кумулятивная функция распределения вероятностей (CDF) лучших значений коэффициентов $\omega_1 \ldots \omega_4$ (слева) и размер буферов (bit) 3-ступенчатого конвейера, оптимизированного HUX (сплошная) и SOX (точечная), а также CDF (пунктирная) в зависимости от размера популяции (справа)

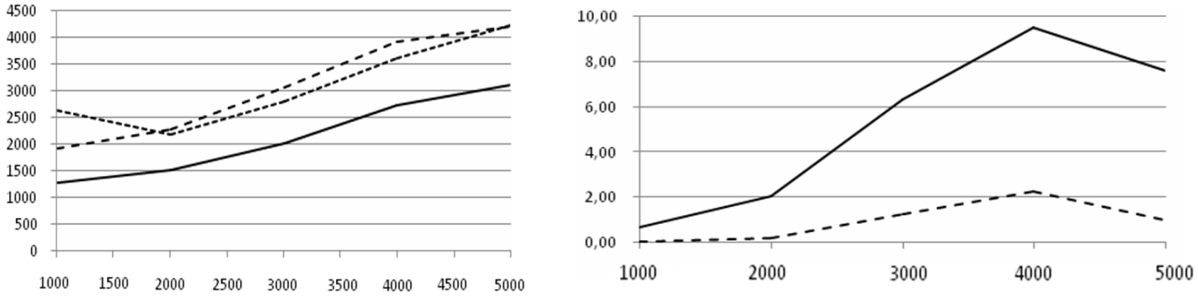

Рис. 7. Размер буферов (bit) для 4-ступенчатого конвейера для GA (сплошная), ASAP (пунктирная) и ALAP (точечная) в зависимости от размера проекта (слева) и среднее время CPU в секундах (сплошная) и его среднеквадратичное отклонение (пунктирная) для алгоритма HDA-GA от размера проекта (справа)

Заключение

Предложен быстрый эвристический генетический алгоритм оптимизации вычислительных конвейеров, минимизирующий размер конвейерных буферов для описаний, включающих тысячи операторов. Генетический алгоритм дает качество решений, близкое к оптимальному, и выигрывает до двух раз по размеру конвейерных буферов у известных алгоритмов нисходящего и восходящего обходов графа потока данных.

Список литературы / References

1. Scheduling and hardware sharing in pipelined data paths / K.S. Hwang [et. al.] // Proc. ICCAD-89, November 1989. P. 24–27.
2. Sun W., Wirthlin M., Neuendorffer S. FPGA pipeline synthesis design exploration using module selection and resource sharing // Trans. Comp.-Aided Des. Integ. Cir. Sys. 2007. Vol. 26, No. 2. P. 254–265.
3. Bakshi S., Gajski D. Component Selection for High-Performance Pipelines // IEEE Trans. VLSI Syst. 1996. Vol. 4, No. 2. P. 181–194.
4. Prihozhy A. High-level Synthesis through Transforming VHDL Models // System-on-Chip Methodologies and Design Languages. 2001. P. 135–146.
5. Dataflow/actor-oriented language for the design of complex signal processing systems / C. Lucarz [et. al.] // Proc. Conf. on Design and Architectures for Signal and Image processing, November 2008. P. 1–8.
6. Synthesis and Optimization of Pipelines for HW Implementations of Dataflow Programs / A. Prihozhy [et. al.] // IEEE Trans. on CAD of Integrated Circuits and Systems. 2016. Vol. 34, No. 10. P. 1613–1626.
7. Rahman H., Prihozhy A., Mattavelli M. Pipeline Synthesis and Optimization of FPGA-Based Video Processing Applications with CAL // EURASIP Journal on Image and Video Processing. 2011. Vol. 19. P. 1–28.

Сведения об авторах

Прихожий А.А., д.т.н., профессор, профессор кафедры программного обеспечения информационных технологий и робототехники Белорусского национального технического университета.

Ждановский А.М., аспирант кафедры программного обеспечения вычислительной техники и автоматизированных систем Белорусского национального технического университета.

Карасик О.Н., аспирант кафедры программного обеспечения вычислительной техники и автоматизированных систем Белорусского национального технического университета.

Маттавелли М., заведующий лабораторией Федерального института технологий, г. Лозанна, Швейцария.

Information about the authors

Prihozhy A.A., D.Sci., professor, professor of software information technologies and robotics department of Belarusian National Technical University.

Zhdanouski A.M., graduate student of software of computer facilities and the automated systems department of Belarusian National Technical University.

Karasik O.N., graduate student of software of computer facilities and the automated systems department of Belarusian National Technical University.

Mattavelli M., head of the laboratory of the Federal Institute of Technology, Lausanne, Switzerland.

Адрес для корреспонденции

220013, Республика Беларусь,
г. Минск, пр. Независимости, д. 65,
Белорусский национальный
технический университет
тел. +375-44-765-94-86;
e-mail: prihozhy@yahoo.com;
Прихожий Анатолий Алексеевич

Address for correspondence

220013, Republic of Belarus,
Minsk, Nezavisimosti ave., 65,
Belarusian National Technical University
tel. + 375-44-765-94-86;
e-mail: prihozhy@yahoo.com;
Prihozhy Anatoliy Alekseevich