УДК 004.3

Д. И. Лазоренко, аспирант
Ин-т проблем моделирования
в энергетике им. Г.Е. Пухова НАН Украины
(Украина, 03164, Киев, ул. Генерала Наумова 15,
тел.: (044) 4240328; e-mail: d.lazorenko@gmail.com)

Трансформация кода программ высокого уровня при синтезе цифровых систем для снижения их энергопотребления

(Статью представил д-р техн. наук В. П. Симоненко)

Предложен графический метод объединения циклов на уровне исходного кода для понижения энергопотребления проектируемых устройств за счет уменьшения обращений к памяти. Приведен алгоритм трансформации кода и примеры, подтверждающие его эффективность. Алгоритм является основой при автоматизации трансформации кода программ на языках высокого уровня.

Запропоновано графічний метод об'єднання циклів на рівні вихідного коду для зниження енергоспоживання пристроїв, що проектуються за рахунок зменшення звернень до пам'яті. Наведено алгоритм трансформації коду і приклади, що підтверджують його ефективність. Алгоритм є основою при автоматизації трансформації коду програм мовами високого рівня.

Ключевые слова: автоматизация проектирования устройств, объединение циклов, снижение энергопотребления.

Развитие полупроводниковых технологий привело к возникновению концепции система-на-кристалле. Сложность современных приложений и использование субмикронных технологий обуславливают необходимость снижения энергопотребления таких систем путем применения оптимальных решений в процессе проектирования.

Современные цифровые системы, например мультимедийные приложения, переносные телефоны, карманные персональные компьютеры, обрабатывают большие массивы данных по сложным алгоритмам. Каждое новое поколение переносных цифровых устройств потребляет энергии значительно больше предыдущего, т. к. в нем реализованы дополнительные возможности, например телевидение или качественная видеокамера в мобильных телефонах. В то же время, увеличение энергоемкости новых

поколений переносных источников питания происходит гораздо более медленными темпами. Таким образом, развитие технологий переносных источников питания не успевает за увеличением энергопотребления новых приложений. Кроме того, пониженное энергопотребление позволяет упростить разводку шин питания на кристалле, приводит к уменьшению шумов на шинах питания и проявления эффекта электромиграции и электромагнитного излучения.

Рассмотрим основные источники энергопотребления в микросхемах и более детально остановимся на методах преобразования кода для уменьшения обращений к памяти.

Источники энергопотребления в схемах КМОП (комплиментарный металл — оксид — полупроводник). Значительное число современных микросхем производится с помощью КМОП технологии, поэтому источники энергопотребления будут рассматриваться применительно к этой технологии. Существует четыре составляющие энергопотребления КМОП схем: токи короткого замыкания, статические токи, паразитные токи утечки и динамическое рассеяние энергии.

Наибольший вклад в энергопотребление вносит динамическое рассеяние энергии. Оно происходит из-за зарядки (разрядки) узлов схемы и может быть представлено следующей формулой:

$$P_{\text{dyn}} = CV_{\text{dd}}^2 \alpha f,$$

где C — суммарная емкость в узлах схемы; V_{dd} — величина напряжения питания; f — частота переключений; α — коэффициент активности переключений (средняя доля логических вентилей, переключаемых за один такт сигнала синхронизации) [1].

На динамическое рассеяние энергии приходится большая часть полных потерь энергии. Переключательная активность в значительной мере определяется программным обеспечением цифровой системы [2,3].

Потенциальный выигрыш в энергопотреблении. В работе [4] обоснован вывод о том, что потенциальный выигрыш в энергопотреблении тем выше, чем выше уровень абстракции процесса проектирования, на котором принимается решение. Для системного уровня выигрыш может составлять от 50 до 90 %, на поведенческом уровне – от 40 до 70 %, на RTL уровне – от 30 до 50 %, на уровне вентилей – от 20 до 30 %, на уровне транзисторов – от 10 до 20 %, на уровне топологии – от 5 до 10 %.

Энергопотребление схем памяти. Современные цифровые системы используют большие объемы памяти. Схемы памяти могут занимать от 50 до 80 % площади полупроводникового кристалла. Известно, что для схем памяти характерны большие паразитные токи утечки. В каждом новом

поколении КМОП технологий уменьшается размер транзисторов и напряжение питания микросхем. Соответственно уменьшается пороговое напряжение у МОП транзисторов, что приводит к увеличению паразитных токов утечки. Так, при уменьшении порогового напряжения на 100 мВ происходит увеличение токов утечки в 10—16 раз. Кроме того, на обращение к памяти тратится много энергии. Например, на операцию чтения из внешней памяти расходуется в 33 раза больше энергии, чем на операцию 16-тибитного сложения.

Согласно прогнозу международной организации International Technology Roadmap for Semiconductors схемы памяти будут занимать все больше площади на полупроводниковых кристаллах: в 2008 г. — 83 %, в 2011 г. — 90 %, в 2014 г. — 94 %. Величина динамического энергопотребления схем памяти в ближайшем будущем также будет увеличиваться. Например, по прогнозу на 2010 г. динамическое рассеяние энергии на схемах памяти увеличится почти в два раза, к 2015 г. — в два с половиной раза, к 2020 г. — в три раза [1, 5]. Согласно тому же прогнозу потери энергии на схемах памяти в результате воздействия паразитных токов утечки будут возрастать.

Таким образом, в процессе проектирования необходимо добиваться уменьшения объема требуемой приложению памяти и числа обращений к ней. Для этого следует оптимизировать систему на поведенческом уровне. Важно, чтобы такая оптимизация проводилась перед разделением системы на программную и аппаратную части, поскольку это позволяет применить однообразный подход к обработке всей памяти. Циклы «for» представляют собой именно ту часть исходного кода приложения, которая ответственна за использование массивов в приложениях, обрабатывающих большие объемы данных по сложным алгоритмам [6—8].

Для сокращения объема требуемой памяти необходимо уменьшить размер и число временных массивов, создаваемых в процессе обработки данных. Уменьшения объема памяти можно также добиться повторным использованием одних и тех же адресов памяти разными массивами.

Уменьшение энергопотребления происходит в результате уменьшения числа обращений центрального процессора (ЦП) к основному запоминающему устройству (ОЗУ) в связи с хранением повторно используемых данных в регистрах ЦП или кэш памяти. В работе [6] утверждается, что элементы памяти, расположенные ближе к вершине ее иерархии, т. е. регистровая память, кэш обладают меньшим размером и потребляют меньше энергии. Утверждается также, что расход энергии на обращение к внешнему ОЗУ больше, чем при обращении к памяти на том же кристалле. В работе [9] приведены результаты эксперимента, свидетельствующие о том, что на обращение к памяти на том же кристалле затрачивается в три

раза меньше энергии, чем на обращение к внешнему ОЗУ. В работе [10] приведен результат оптимизации алгоритмического описания MPEG-4 приложения на языке С (а именно преобразования циклов для повторного использования данных) и получен выигрыш в энергопотреблении приблизительно в пять раз.

Воспользуемся формулой для динамического рассеяния энергии. Сравним расход энергии на обращение к кэш памяти, которая по определению меньше ОЗУ (внешнее оно или расположено на том же кристалле). Поскольку кэш память меньше ОЗУ, разрядность шины данных у нее должна быть меньше, чем у ОЗУ. Это значит, что управляющих логических схем в кэш памяти меньше, чем в ОЗУ. Поэтому коэффициент переключательной активности α будет иметь меньшее значение в случае работы с кэш памятью (предполагаем, что напряжение питания будет одинаковым для всех видов памяти). По той же причине величина C должна быть меньше для кэш памяти. Если же ОЗУ — внешнее, то в величину C включается немалая емкость соединений на печатной плате. Предполагая, что величина f также одинакова для обоих видов памяти, в соответствии с формулой для P_{dyn}, определяем, что на обращение к кэш памяти будет расходоваться меньше энергии.

Рассмотрим простой пример преобразований циклов в тексте программы, приводящих к уменьшению размера памяти и числа обращений к ней.

На рис. 1, *а* представлен исходный код приложения, в котором обрабатываются четыре массива: **a**, **b**, **c**, и **d**, на рис. 1, *б* — результаты объединения циклов. Легко заметить, что в полученном цикле элемент массива **b** используется сразу же после его вычисления. Это дает возможность хранить его значение в быстрой памяти, которая менее энергоемка, и поэтому не требуется повторно считывать элемент из ОЗУ. Элемент массива **a** читается из ОЗУ только один раз. Таким образом, сокращается число обращений к ОЗУ. Если же предположить, что массив **b** нигде более в программе не используется, то его можно заменить переменной, вследствие чего будет уменьшен объем необходимой памяти [6].

Преобразования циклов в исходном тексте описания приложения. Важность преобразования циклов в исходном тексте программ была понята достаточно давно [11, 12]. Большинство работ по методам преобразования циклов направлены на улучшение распараллеливания вычислений. Эффективность распараллеливания, однако, лишь частично связана с локальностью доступа к данным [13—18]. Во многих работах преобразования циклов проводятся с использованием графов [6, 7, 17].

На рис. 2 каждому выражению из тела цикла S1, S2, S3 и S4 соответствует вершина графа. Ребра графа отображают связи между выраже-

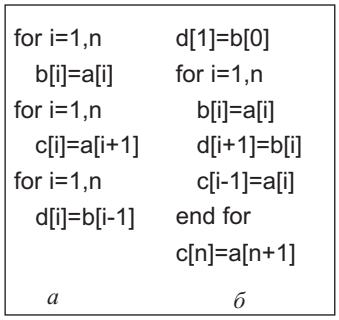


Рис. 1

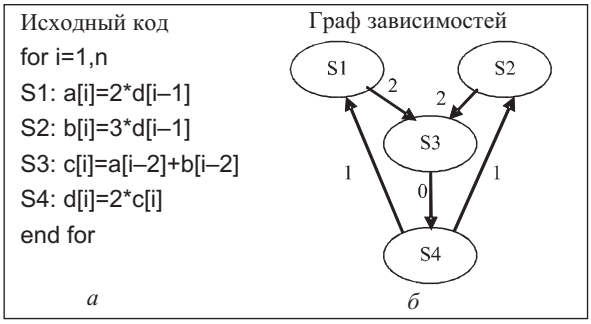


Рис. 2

ниями. Каждому ребру присваивается определенный вес, равный числу итераций между двумя обращениями к одному и тому же массиву. Например, в выражениях S2 и S3 используется массив **b**, в S2 – это элемент b[i], в S3 — b[i–2]. Число итераций между двумя обращениями к массиву **b** равно 2, поэтому ребру, связывающему вершины S2 и S3, присваивается вес 2.

Число ячеек в быстрой памяти, необходимое для хранения данных, относящихся к определенному выражению, определяется весом w_e всех исходящих ребер e соответствующей вершины графа u. Для этого вводится понятие цены вершины, которое определяется всеми исходящими из данной вершины ребрами:

$$C_u = \max_{e=(u,v)\in E} w_e(u^e \to v).$$

Здесь v — вершина графа; E — множество всех ребер графа. Пусть V — множество всех вершин графа. Тогда полное число ячеек быстрой памяти для цикла определяется следующим выражением: $\text{Cost}\ (G) = \sum_{u\in V} C_u$. Минимизировать величину Cost (G) можно, используя метод перенастройки. При этом каждой вершине u ставится в соответствие параметр перенастройки r_u (целочисленная величина). Данный параметр представляет собой величину изменения веса ребра графа. Для перенастроенного графа вес ребра можно тогда записать в виде

$$w_{r,e} = w_e + r_v - r_u, (u^e \to v),\ w_{r,e} \ge 0,\ \forall e \in E.$$

Задача минимизации Cost (G) сводится к задаче целочисленного линейного программирования:

$$\min \sum_{u\in V} C_u,\ w_{r,e} \ge 0,\ C_u \ge w_{r,e},\ \forall e \in E.$$

Данную задачу можно решить с помощью существующего программного обеспечения для задач этого класса, например пакетов LINGO или GAMS. В данном случае выполнялась минимизация размера быстрой памяти.

Графический метод объединения циклов. Как упомянуто выше, основным источником энергопотребления схем памяти являются операции чтения и записи. На рис. 1 показано, что операции объединения и выравнивания циклов позволяют преобразовать исходный код приложения так, чтобы снизить количество обращений к памяти. Рассмотрим графический метод анализа одномерных циклов относительно возможности их объединения. Предлагаемый способ представляет собой неформализованный метод, являющийся лишь первым шагом на пути создания формализованного метода, способного автоматически преобразовывать исходный текст описания реальных приложений.

На рис. 3 графически отображены связи между элементами двух массивов: **a** и **b**. Горизонтальной стрелкой показана ось итераций, на которой не указано начало координат, поскольку для предлагаемой процедуры анализа и преобразования циклов это не требуется. Штриховыми линиями обозначены отдельные итерации вычислений. Ниже оси точками представлены элементы массивов **a** и **b**. Вертикальными стрелками указаны связи между элементами массивов в процессе вычислений. Например, элемент **a** [1] используется при вычислении элемента **b** [1], поэтому стрелка выходит из точки, соответствующей **a** [1], и заканчивается в точке, соответствующей **b** [1].

Очевидно, данные циклы могут быть объединены. В результате хранения одного вычисленного значения элемента **a** [i] в быстрой памяти получаем выигрыш в четыре операции чтения из медленной памяти. В данном случае стрелки, связывающие элементы двух массивов, совпадают со штриховыми линиями итераций, что графически отображает необходимость хранить одно значение элемента массива **a** в процессе выполнения одной итерации.

Рассмотрим следующий пример. Представленные на рис. 4 два цикла также могут быть объединены. Легко заметить, что для уменьшения числа операций чтения из медленной памяти необходимо хранить в быстрой памяти уже два значения элементов массива: **a** [i] и **a** [i–1]. Как видим, на одной штриховой линии итерации находится точка, из которой выходит стрелка, и точка, в которой заканчивается стрелка, связывающая элементы двух массивов на разных линиях итерации. Следовательно для расчета элементов массива **b** используются только три элемента массива **a**, и необходимо также задать значение элемента **b** [0] вне тела цикла (стрелка с основанием).

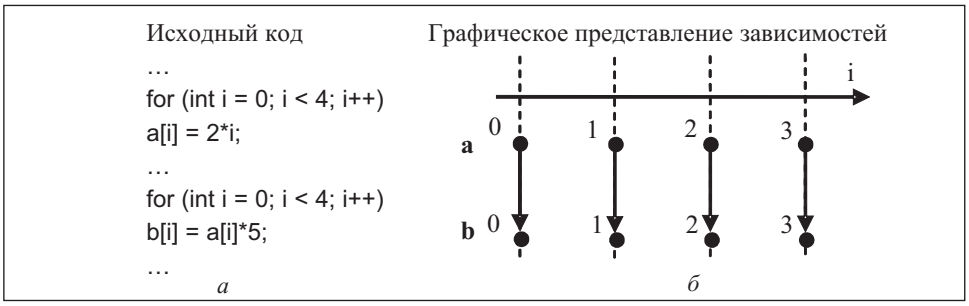


Рис. 3

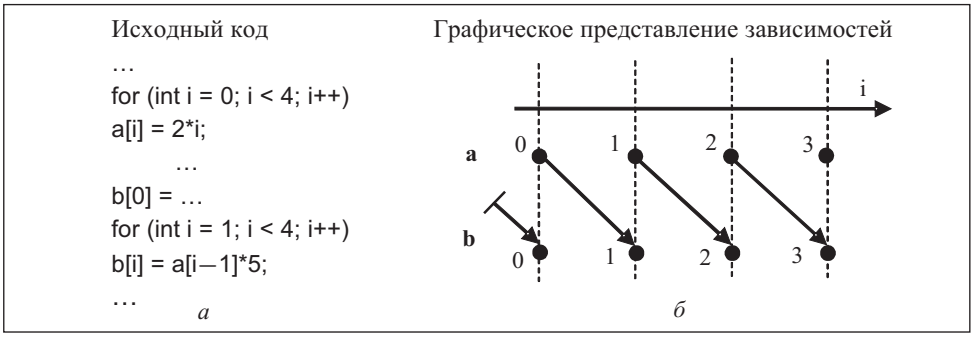


Рис. 4

Если над циклами провести операцию выравнивания, то код и соответствующее ему графическое отображение будут выглядеть так, как показано, на рис. 5. Очевидно, что рис. 5, *б* может быть легко получен из рис. 4, *б* смещением элементов массива **b** влево на расстояние, соответствующее одной итерации, с сохранением связей между элементами массивов. Операция формирования трансформированного кода из преобразованного графического представления следующая. Поскольку существуют только связи a [0] → b [1], a [1] → b [2], a [2] → b [3], логично было бы сформировать новый цикл, выполняющий вычисления только над этими элементами, а вычисления элементов a [3] и b [0] провести вне тела цикла. Тогда значения итерационной переменной i могут быть 0, 1, и 2. Видно, что при наличии связей a [0] → b [1], a [1] → b [2], a [2] → b [3] в теле нового цикла, a [i]-му элементу соответствует b [i+1]-й элемент.

Рассмотрим следующий пример. В [7] утверждается, что циклы, представленные на рис. 6, не могут быть объединены, поскольку нельзя использовать еще не вычисленное значение элемента a [i +1]. Как видно из рис. 6, *б*, наличию подобной ситуации графически соответствует наличие стрелок с отрицательной проекцией на ось итераций.

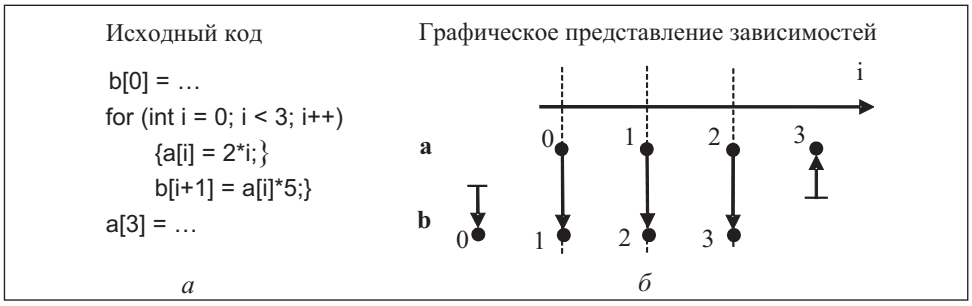


Рис. 5

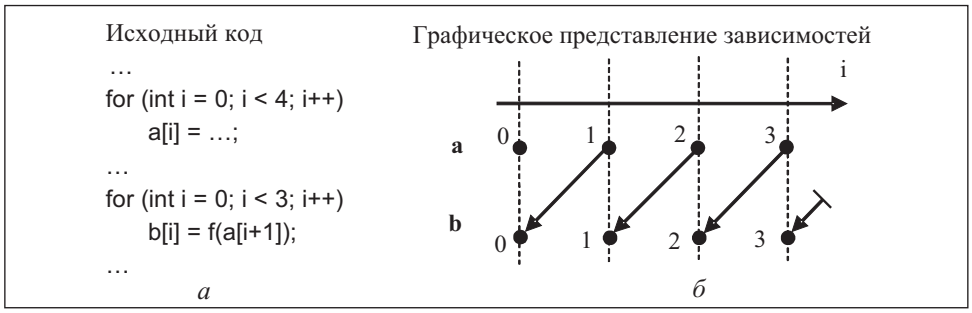


Рис. 6

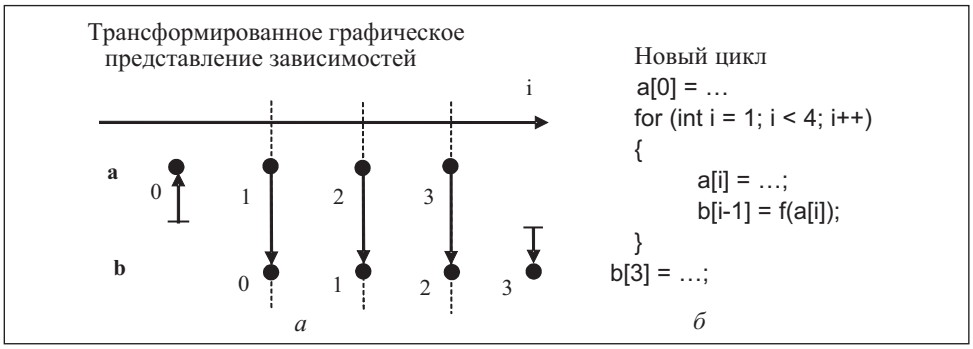


Рис. 7

Однако, используя предлагаемый подход, объединить такие циклы возможно, как это показано на рис. 7, *а*, где представлено преобразованное графическое отображение связей между элементами циклов, приведенных на рис. 6, *а*. Точки, соответствующие элементам массива **b**, были сдвинуты вправо на расстояние одной итерации. Преобразованное отображение не содержит более стрелок с отрицательной проекцией на ось итераций. Видно, что существуют только связи a [1] → b [0], a [2] → b [3], a [3] → b[2].

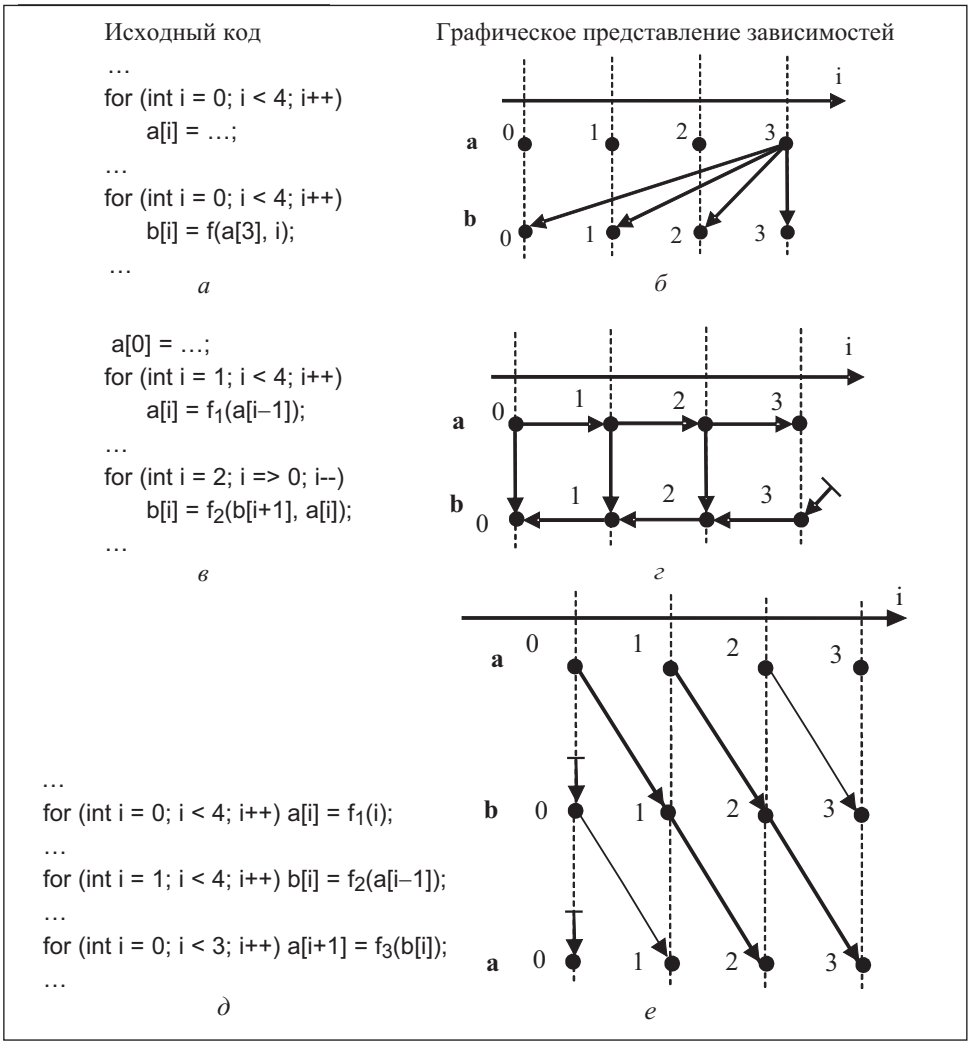


Рис. 8

Тогда формируем новый цикл, выполняющий вычисления только над этими элементами, а вычисления элементов a [0] и b [3] проводим вне тела цикла. При этом значения итерационной переменной i могут быть 1, 2, и 3. Легко заметить, что для связей a [1] → b [0], a [2] → b [3], a [3] → b [2] в теле нового цикла a [i]-му элементу соответствует b [i–1]-й элемент. Код объединенного цикла представлен на рис. 7, *б*.

Циклы невозможно объединить тогда, когда их графическое представление нельзя преобразовать сдвигом точек, соответствующих элементам массивов разных циклов, таким образом, чтобы более не существовало

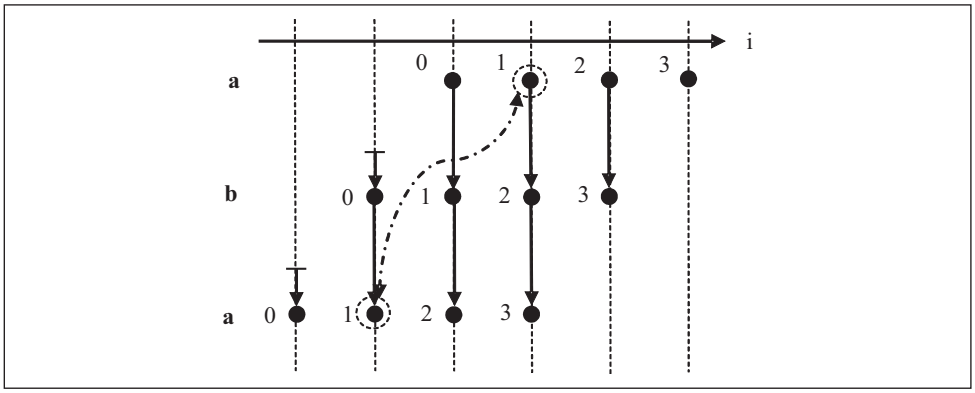


Рис. 9

стрелок с отрицательными проекциями на ось итераций. Примеры таких циклов приведены на рис. 8.

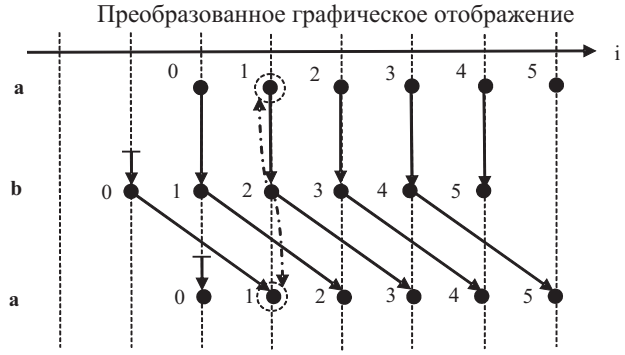
Как видно на рис. 8, *б*, можно сдвигать точки, соответствующие элементам массива **b**, вправо до тех пор, пока все проекции всех стрелок станут положительными. Однако тогда только элементы a [3] и b [0] будут расположены на одной линии итерации, т. е. говорить о каком-либо цикле будет невозможно.

На рис. 8, *в* видно, что в данных циклах есть связи между элементами одного и того же массива, а на рис. 8, *г* есть стрелки с отрицательной проекцией на ось итераций. Легко заметить, что данное графическое отображение нельзя преобразовать так, чтобы избавиться от стрелок с отрицательной проекцией на ось итераций. Поэтому данные два цикла не могут быть объединены.

В примерах рассмотрены циклы со связями по данным. Теперь проанализируем возможность объединения циклов, в которых присутствуют связи по выходу и антизависимости.

На рис. 8, *д* представлен некоторый исходный код, в котором присутствуют циклы со связями по данным (циклы первый и второй, а также второй и третий), по выходу (циклы первый и третий) и антизависимось (циклы второй и третий), а на рис. 8, *е* — графическое отображение этого исходного кода.

На рис. 9 представлено графическое отображение связей между элементами циклов, преобразованное путем смещения влево точек, соответствующих массиву **b**, на одну итерацию, а также смещением влево точек, соответствующих массиву **a** в третьем цикле, на две итерации. Нетрудно заметить, что все циклы объединить в один нельзя, так как значения элементов массива **a** перезаписываются на более поздних итера-



Исходный код

```
…
for ( i=0; i<6; i++) a[i] = f1(i);
…
b[0] = …;
for (i = 1; i<6; i++) b[i] = f2(a[i–1]);
…
a[0] = …;
for (i = 0; i<5; i++) a[i+1] = f3(b[i]);
…
```

Код объединенных циклов

```
b[0] = …;
b[1] = …;
a[0] = …;
for (i = 1; i < 5; i++)
{
  a[i] = f1(i);
  b[i+1] = f2(a[i]);
  a[i] = f3(b[i-1]);
}
a[5] = …;
```

Рис. 10

циях (это обозначено штрих-пунктирной стрелкой). Легко заметить также, что могут быть объединены первый и второй циклы, либо второй и третий.

Если организовать вычисления так, чтобы один и тот же элемент массива **a** рассчитывался на одной линии итерации, то все три цикла можно объединить. Такое преобразованное графическое отображение циклов и код объединенного цикла представлены на рис. 10.

На рис. 11, *а*, *б* представлены некоторый исходный и соответствующий ему модифицированный коды, полученные методом объединения циклов, описанных в [7]. На рис. 11, *в* изображено графическое представление связей между элементами массивов.

Как видно из рис. 11, *б*, способ, предлагаемый в [7], позволяет объединить только три цикла из пяти. Теперь проведем объединение циклов способом, предлагаемым в данной работе. Как видно из рис. 11, *в*, точки, соответствующие элементам массивов **a2** и **a3**, связывают стрелки, которые имеют отрицательные проекции на ось итераций. Смещением влево точек, соответствующих элементам массива **a2**, необходимо эти стрелки сделать перпендикулярными оси итераций. Смещение выполняется на

Исходный код

```
…
for (int i = 0; i<6; i++) a1[i] = f1(i);
…
for (int i = 0; i<6; i++) a2[i] = f2(a1[i]);
…
for (int i = 0; i<5; i++) a3[i] = f3(a2[i+1]);
…
for (i = 1; i<6; i++) a4[i] = f4(a1[i-1],a3[i]);
…
for (i = 0; i<6; i++) a5[i] = f5(a2[i],a3[i],a4[i]);
…
```

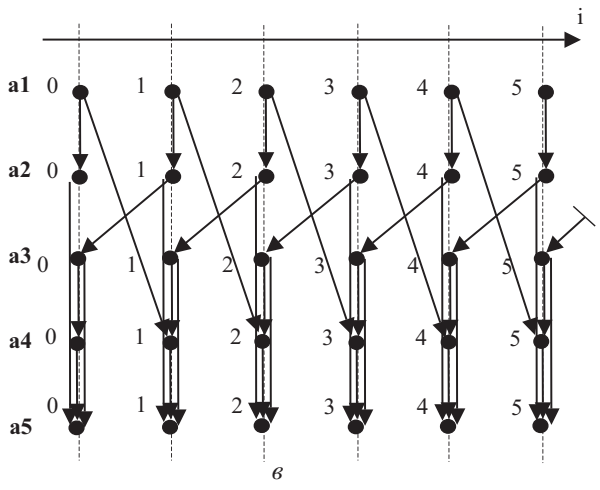
а

Модифицированный код

```
…
for (int i = 0; i<6; i++) a1[i] = f1(i);
…
for (int i = 0; i<6; i++) a2[i] = f2(a1[i]);
…
for (int i = 1; i<5; i++)
{
  a3[i] = f3(a2[i+1]);
  a4[i] = f4(a1[i-1],a3[i]);
  a5[i] = f5(a2[i],a3[i],a4[i]);
}
…
```

б

Графическое отображение исходного код

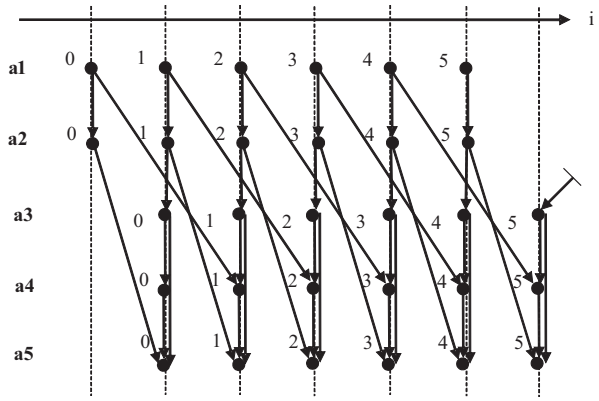


в

Рис. 11

расстояние, соответствующее одной итерации. После этого проекции стрелок, связывающих точки, соответствующие элементам массивов **a1** и **a2**, станут отрицательными. Для устранения этого, проводим смещение влево на расстояние, соответствующее одной итерации, точек, представляющих элементы массива **a1**. Теперь данные стрелки также станут перпендикулярными оси итераций. В результате получаем графическое отображение, представленное на рис. 12, *а*. В нем не осталось стрелок с отрицательными проекциями на ось итераций, поэтому все пять циклов могут быть объединены в один.

Преобразованное графическое отображение



а

Исходный код

```
…
for ( i=0; i<6; i++) a1[i] =f1(i);
for (i = 0; i<6; i++) a2[i] = f2(a1[i]);
for (i = 0; i<5; i++) a3[i] = f3(a2[i+1]);
a3[5] = …;
for (i = 1; i<6; i++) a4[i] = f4(a1[i–1],a3[i]);
a4[0] = …;
for (i = 0; i<6; i++) a5[i] =
f5(a2[i],a3[i],a4[i]);
…
```

б

Модифицированный код

```
…
a1[0] = ...;
a2[0] = ...;
for (i = 1; i<6; i++) {
   a1[i] = f1(i);
   a2[i] = f2(a1[i]);
   a3[i–1] = f3(a2[i]);
   a4[i–1] = f4(a1[i–2],a3[i–1]);
   a5[i–1] = f5(a2[i–1],a3[i–1],a4[i–1]);
}
a3[5] = ...;
a4[5] = ...;
a5[5] = ...;
…
```

в

Рис. 12

Из рис. 12, *а* видно, что цикл, объединяющий исходные пять, можно сформировать из элементов массивов a1[1..5], a2 [1..5], a3 [0..4], a4 [0..4], a5 [0..4]. Вычислять значения элементов a1[0], a2 [0], a3 [5], a4 [5], и a5 [5] следует проводить вне тела нового цикла. Значения итерационной переменной i будут 1, 2, 3, 4, 5. Из рис. 12, *а* легко заметить, что на линии одной итерации располагаются элементы a1[i], a2 [i], a3 [i–1], a4 [i–1], a5 [i–1]. Код нового, объединенного цикла, приведен на рис. 12, *в*.

Рассмотрим исходный код, показанный на рис. 12, *б*. В процессе его выполнения происходит 30 операций записи в медленную память и 39

операций чтения из медленной памяти. При выполнении соответствующего ему улучшенного кода, представленного на рис. 12, *в* происходит 30 операций записи в медленную память и четыре операции чтения из медленной памяти. Таким образом, экономится энергия на 35 операций чтения из медленной памяти. При этом в быстрой памяти должны храниться семь элементов массивов a1[i], a1[i–1], a1[i–2], a2 [i], a2 [i–1], a3 [I–1], a4 [i–1]. Если предположить, что a2, a3, a4 — это массивы для хранения промежуточных данных, которые более нигде в коде не требуются, то можно сократить объем необходимой приложению медленной памяти, сохраняя значения a2 [i], a2 [i–1], a3 [i–1], a4 [i–1] в скалярных переменных в быстрой памяти. Число операций записи в медленную память сократиться при этом на 20.

В работе [19] приведены данные об энергопотреблении при операциях обращения к памяти. Так, на операцию чтения расходуется 0,09 мВ, на операцию записи – 0,17 мВ. Используя эти данные и считая, что все повторно используемые данные из приведенного выше примера можно хранить в регистрах ЦП, получаем следующее. На операции чтения в исходном варианте записи программы затрачивается 39·0,09 мВ = 3,51 мВ, на операции чтения в преобразованном согласно предлагаемому способу варианте текста — 4 · 0,09 мВ = 0,36 мВ. Таким образом, выигрыш в энергопотреблении составляет 3,15 мВ. Для операций записи в память выигрыш может составлять 20·0,17 = 3,4 мВ. Следовательно, в общем потенциальный выигрыш в энергопотреблении для приведенного примера может составлять 6,55 мВ или 4,2 раза.

Для обобщения приведем алгоритмы построения и преобразования графического отображения циклов, а также формирования нового кода по графическому отображению.

А л г о р и т м построения графического отображения.

1. На оси итераций произвольно выбираем расстояние, называемое расстоянием одной итерации.

2. Ниже оси итераций рисуем точки, изображающие элементы массивов, на расстоянии одной итерации одна от другой. Массивы отображаются в том порядке, в котором выполняются вычисления над ними в программе.

3. Между точками проводим стрелки. Каждая стрелка начинается в точке, представляющей элемент, данные которого используются при вычислении другого элемента, и заканчивается в точке, представляющей этот другой элемент.

А л г о р и т м преобразования графического представления циклов.

1. Определяем знак проекции всех стрелок.

2. Если существуют стрелки с отрицательными проекциями, то смещением точек, связанных такими стрелками, вдоль или против оси итераций

делаем данные стрелки либо перпендикулярными оси итераций, либо делаем их проекцию на ось итераций положительной.

3. Если никакими преобразованиями нельзя убрать стрелки с отрицательными проекциями на ось итераций, то все исследуемые циклы объединить невозможно.

4. Если все циклы не могут быть объединены, разбиваем циклы на группы, которые расположены выше и ниже стрелок с отрицательными проекциями на ось итераций.

4.1. Если внутри какой-либо группы отсутствуют циклы со связями по выходу, то все циклы в каждой группе могут быть объединены.

4.2. Если внутри какой-либо из групп есть циклы со связями по выходу, то смещением точек, соответствующих данной группе циклов, добиваемся, чтобы точки циклов со связью по выходу с одинаковыми номерами находились на одной линии итерации. Во время этой операции не должна появиться стрелка с отрицательными проекциями на ось итераций.

4.3. Если невозможно добиться требуемого в предыдущем пункте результата, то циклы со связями по выходу не могут быть объединены, а значит, невозможно объединить все циклы в данной группе. Циклы, расположенные между циклами со связями по выходу, следует объединять с каким-либо из этих циклов в зависимости от того, в каком случае можно добиться большего числа вертикальных стрелок связей между циклами, которые можно объединить.

А л г о р и т м формирования нового кода в соответствии с преобразованным графическим представлением циклов.

1. Коды циклов, которые не могут быть объединены с другими, оставить в первоначальном виде.

2. В графическом отображении выбрать те группы циклов, которые могут быть объединены. Для каждой группы циклов, которые могут быть объединены, выполнять следующие операции.

2.1. Для группы циклов, которые могут быть объединены, выбрать область графического отображения, в которой на одной линии итераций находятся все элементы массивов данной группы циклов.

2.2. В найденной области выбрать только те линии итераций, которым присущи одинаковые связи, представленные стрелками. Элементы, расположенные на выбранных линиях, будут вычисляться в теле нового цикла. Все элементы массивов, не находящиеся на этих линиях итераций, следует вычислять вне тела нового цикла в соответствии с существующими связями.

2.3. В найденной области, определить элементы массива, с которых начинаются вычисления. Затем определить минимальный и максимальный номера таких элементов. Эти номера задают границы значений итерационной переменной объединенного цикла.

2.4. Для одной из выбранных линий итерации вычислить разницу между номером элемента массива, выбранного на предыдущем шаге алгоритма, и всеми другими элементами на линии итерации.

2.5. В теле объединенного цикла переписать формулы вычислений, заменив индексные выражения следующим образом.

2.5.1. Элементы, с которых начинаются вычисления, получают новое индексное выражение, равное новой итерационной переменной.

2.5.2.Все элементы вычислять, используя индексные выражения вида «индексная переменная плюс разница», вычисленная на шаге 2.4 данного алгоритма.

Выводы. Предлагаемый графический подход представляет собой простой и наглядный способ анализа циклов относительно возможности их объединения. Как показано на примере, данный способ позволяет добиться лучших результатов в объединении циклов, чем, например, предлагаемый в [7]. Данный способ позволяет также анализировать влияние связей внутри одного цикла на возможность объединения его с другим, что отсутствует в методе, предложенном в [7]. В результате объединения циклов, выполненного согласно предлагаемому методу, можно существенно сократить число операций обращения к медленной памяти, а также уменьшить ее размер.

Предлагаемый способ представляет собой неформализованный метод, рассматриваемый как первый этап разработки формализованного метода, способного объединять многомерные циклы в тексте описания реальных приложений. Планируется рассмотреть вопросы автоматизации данного метода для ввода его в процесс оптимизации кода компилятором с языка SystemC. Интерес представляет также применение данного подхода к проблеме уменьшения требуемой быстрой памяти для случая вычислений внутри одного цикла [6].

The graphic method of loops grouping is proposed on the level of original code. The method is used for reducing of energy consumption by devices projected at the expense of memory access reduction. The code transformation algorithm is offered. Examples are cited, which confirm its effectiveness. The algorithm is a basis for automation of code-to-code transformation on the high level languages.

1. *Veendrick H. J .M.* Deep-Submicron CMOS ICs. — From Basics to ASICs. — Dordecht: Kluwer academic publishers, 2000. — 539 p.
2. *Poppen F.* Low Power Design Guide. — OFFIS Research Institute. Http://www.offis.de, 2000. — 18 p.
3. *Kim H. S., Irwin M. J., Vijaykrishnan N., Kandemir M.* Effect of compiler optimizations on memory energy// IEEE Workshop on Signal Processing Systems. — Piscataway: Institute of Electrical and Electronics Engineers, 2000. — P. 663 — 672.
4. *Sproch J.* High Level Power Analysis and Optimization. — Tutorial//International Symposium on Low Power Electronics and Design. — N.Y.: ACM Press, 1997. — 10 p.

5. *International* Technology Roadmap for Semiconductors. http://public.itrs.net/
6. *Fraboulet A., Huard G., Mignotte A.* Loop Alignment for Memory Accesses Optimization, Twelfth International Symposium on System Synthesis. — Piscataway: IEEE Computer Society Press, 1999. — P. 71 — 77.
7. *Fraboulet A., Kodary K., Mignotte A.* Loop fusion for memory space optimization: The 14th International Symposium on System Synthesis. — Piscataway: IEEE Computer Society Press, 2001. — P. 95 — 100.
8. *Catthoor F., Franssen F., Wuytack S., Nachtergaele L., De Man H.* Global communication and memory optimizing transformations for low power signal processing systems//Workshop on VLSI Signal Processing, VII. — Piscataway: Institute of Electrical and Electronics Engineers, 1994. — P. 178 — 187.
9. *Fraboulet A., Just-Meunier L., Mignotte A.* Memory Optimization of Data Flow Applications at the Codesign Level// Cadence Techn. Conf. — San Jose: Cadence Design Systems, 2001. — P. 165 — 172.
10. *Brockmeyer E., Catthoor F., Bormans J., De Man H.* Code transformations for reduced data transfer and storage in low power realisation of MPEG-4 full-pelmotion estimation//IEEE Intern. Conf. on Image Processing.— Piscataway: Institute of Electrical and Electronics Engineers, 1998. — P. 985— 989.
11. *Banerjee U., Eigenmann R., Nicolau A., and Padua D. A.* Automatic program parallelization // Proceedings of the IEEE.— 1993. — Vol. 81, No 2. — Piscataway: Institute of Electrical and Electronics Engineers. — P. 211—243.
12. *Bacon D., Graham S., Sharp O.* Compiler transformations for high-performance computing// ACM Computing Surveys. — 1994. — Vol. 26, No 4. — N.Y.: ACM Press.— P. 345—420.
13. *Wolfe M.* The tiny loop restructuring tool// Proc.of 1991 Intern. Conf. on Parallel Processing. — Boca Raton: CRC Press, 1991. — P. 46—53.
14. *Wolfe M.* High-Performance Compilers for Parallel Computing. — Boston: Addison-Wesley, 1996. — 500 p.
15. *Wolfe M.* Optimizing Supercompilers for Supercomputers. — Cambridge: MIT Press, 1989. — 176 p.
16. *Darte A., Risset T., Robert Y.* Loop nest scheduling and transformations// Environments and Tools for Parallel Scientific Computing. — Amsterdam: Elsevier Science Publishers, 1993. — P. 309—332.
17. *Darte A.* On the complexity of loop fusion//Intern. Conf. on Parallel Architectures and Compilation Techniques. — Piscataway: IEEE Computer Society Press, 1999. — P. 149—157.
18. *Panda P., Catthoor F., Dutt N., at al.* Data and Memory Optimization Techniques for Embedded Systems//ACM Trans. on Design Automation for Embedded Systems.— 2001. — Vol. 6, No.2. — N.Y.: ACM Press, 2001. — P.142—206.
19. *Meng T. H., Gordon B., Tsern E., Hung A.* Portable video-on-demand in wireless communication//Special issue on «Low power electronics» of the Proceedings of the IEEE. — 1995. — Vol. 83, No. 4. — Piscataway: Institute of Electrical and Electronics Engineers, 1995. — P. 659 — 680.

Поступила 06.11.06;
после доработки 05.03.07

ЛАЗОРЕНКО Дмитрий Иванович, аспирант Ин-та проблем моделирования в энергетике им. Г. Е. Пухова НАН Украины. В 1994 г. окончил Московский государственный ин-т электронной техники (Технический университет). Область научных исследований — снижение энергопотребления цифровых систем способом высокоуровневых трансформаций исходного описания приложения на языках высокого уровня C/C++, SystemC, VHDL.