УДК 004.3

**Д. И. Лазоренко,** канд. техн. наук,
**А. А. Чемерис,** канд. техн. наук, **В. В. Тарапата**, аспирант
Ин-т проблем моделирования
в энергетике им. Г. Е. Пухова НАН Украины
(Украина, 03164, Киев, ул. Генерала Наумова, 15,
тел.(044) 424-0328, E-mail: d.lazorenko@gmail.com)

# Снижение энергопотребления цифровых устройств с помощью операции объединения циклов

Предложен способ объединения многомерных циклов на уровне исходного текста описания цифровой системы для снижения ее энергопотребления в результате уменьшения числа обращений к основному запоминающему устройству. Алгоритм является основой для автоматизации трансформации текста программ на языках высокого уровня.

Запропоновано спосіб об'єднання багатовимірних циклів на рівні вихідного тексту опису цифрової системи для зниження її енергоспоживання в результаті зменшення кількості звернень до основного запам'ятовуючого пристрою. Алгоритм є основою для автоматизації трансформації тексту програм на мовах високого рівня.

*К л ю ч е в ы е  с л о в а: автоматизация проектирования устройств, объединение циклов, снижение энергопотребления.*

Развитие полупроводниковых технологий привело к тому, что энергопотребление стало важным параметром при проектировании современных цифровых систем. Это верно не только для переносных устройств, например мобильных телефонов, но и для стационарных приложений, поскольку с уменьшением размера транзисторов плотность рассеиваемой на полупроводниковых кристаллах мощности возрастает до таких значений, которые требуют применения специальных технологий охлаждения и дорогих корпусов микросхем.

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, так как 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 отстает от темпов увеличения энергопотребления новых приложений. Кроме того, малое энергопотребление позволяет упростить разводку шин питания на кристалле, приводит к уменьшению шумов на шинах питания, а также проявления эффекта электромиграции и электромагнитного излучения.

Энергопотребление является важным параметром при проектировании встроенных систем. Встроенные системы — это компьютерные системы, в которых компьютер обычно встроен в управляемое им устройство. Поскольку встроенная система предназначается для выполнения строго определенных задач, это позволяет разработчикам, имея полную информацию о выполняемой задаче, оптимизировать встроенную систему по энергопотреблению.

**Источники энергопотребления в КМОП схемах.** Подавляющее число современных микросхем выполняется с помощью КМОП (комплементарный металл—оксид—полупроводник) технологии. Поэтому источники энергопотребления будем рассматривать 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, которое происходит вследствие 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; $V_{\text{dd}}$ — напряжение питания; $f$ — частота переключений; $\alpha$ — коэффициент активности переключений (средняя доля логических вентилей, переключаемых за один такт сигнала синхронизации) [1].

Большая часть полных потерь энергии приходится на динамическое рассеяние энергии. Переключательная активность в значительной мере определяется программным обеспечением цифровой системы [2].

**Потенциальный выигрыш в энергопотреблении.** В работе [3] обоснован вывод о том, что потенциальный выигрыш в энергопотреблении тем выше, чем выше уровень абстракции этапа проектирования, на котором принимается решение. На системном уровне выигрыш может быть от 50 до 90 %, на поведенческом уровне — от 40 до 70 %, на уровне RTL (Register Transfer Level) — 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 схемы памяти имеют большие паразитные токи утечки. С переходом на каждое новое поколение КМОП технологий уменьшается размер транзисторов и напряжение питания микросхем. Соответственно уменьшается пороговое напряжение у КМОП 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 затрачивается много энергии, например на операцию чтения из внешней памяти расходуется в 33 раза больше энергии, чем на операцию 16-битного сложения. Согласно прогнозу международной организации International Technology Roadmap for Semiconductors схемы памяти будут занимать все больше площади на полупроводниковых кристаллах: в 2011 г. — 90 %, в 2014 г. — 94 %. Величина динамического энергопотребления схем памяти в ближайшем будущем будет также увеличиваться. Например, по прогнозу на 2010 г. динамическое рассеяние энергии на схемах памяти будет в два раза больше, чем на логических схемах, к 2015 г. эта величина достигнет 2,5 раза, к 2020 г. она увеличится до трех раз [4]. Согласно тому же прогнозу потери энергии на схемах памяти за счет паразитных токов утечки будут возрастать.

Таким образом, в процессе проектирования следует добиваться уменьшения объема требуемой для данного приложения памяти и числа обращений к ней.

Проектирование современных цифровых систем начинается с описания системы на языках высокого уровня, например C/C++, SystemC, VHDL, Verilog. В соответствии с изложенным наибольший выигрыш в энергопотреблении можно получить, если оптимизировать исходное описание системы.

Циклы «for» представляют собой именно ту часть исходного текста приложения, которая ответственна за использование массивов в приложениях, а следовательно, за обращения к памяти [5, 6].

Для уменьшения 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 вследствие уменьшения числа обращений к памяти происходит при уменьшении числа обращений центрального процессора (ЦП) к основному запоминающему устройству (ОЗУ) в случае хранения повторно используемых данных в регистрах ЦП или кэш

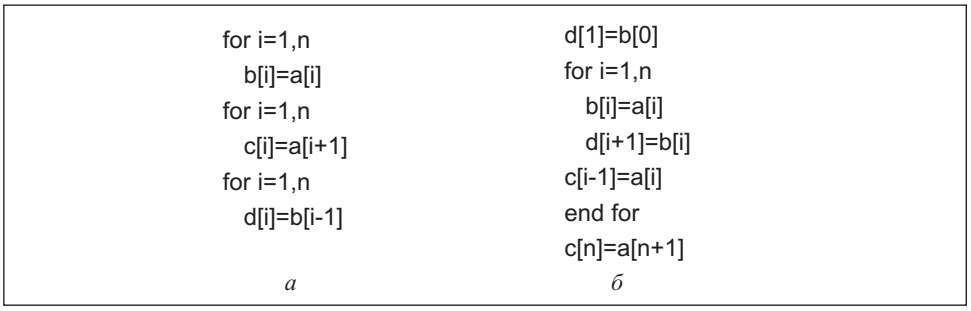



*Рис. 1*

памяти. Так, в работе [5] утверждается, что элементы памяти, расположенные ближе к вершине иерархии памяти, т.е. регистровая память, кэш, имеют меньшие размеры и потребляют меньше энергии, а также что расход энергии на обращение к внешнему ОЗУ больше, чем при обращении к памяти на том же кристалле.

Воспользуемся формулой для динамического рассеяния энергии. Сравним расход энергии на обращение к кэш памяти, которая по определению меньше ОЗУ независимо от того, внешнее оно или расположено на том же кристалле. Поскольку кэш память меньше ОЗУ, разрядность шины данных у нее должна быть меньше, чем у ОЗУ, а это значит, что управляющих логических схем в кэш памяти меньше, чем в ОЗУ. Поэтому коэффициент переключательной активности $\alpha$ будет иметь меньшее значение в случае работы с кэш памятью (при условии, что напряжение питания будет одинаковым для всех видов памяти). По этой же причине величина $C$ должна быть меньше для кэш памяти. Если ОЗУ — внешнее, то величина $C$ содержит значительную емкость соединений на печатной плате. Предполагая, что значения $f$ также одинаковы для обоих видов памяти, по формуле (1) определяем, что на обращение к кэш памяти расходуется меньше энергии.

Простой пример преобразований циклов в тексте программы, которые приводят к уменьшению размера памяти и числа обращений к ней, представлен на рис. 1, где *а* — исходный код приложения, в котором обрабатываются четыре массива: a, b, c и d; *б* — результаты объединения циклов. Легко заметить, что в полученном цикле элемент массива b используется сразу после его вычисления, что дает возможность хранить его значение в быстрой памяти (менее энергоемкой), и поэтому не требуется повторно считывать элемент из ОЗУ. Элемент массива a читается из ОЗУ только один раз, следовательно, сокращается число обращений к ОЗУ. Если предположить, что массив b нигде более в программе не используется, то его можно заменить переменной, что уменьшает объем необходимой памяти [5].

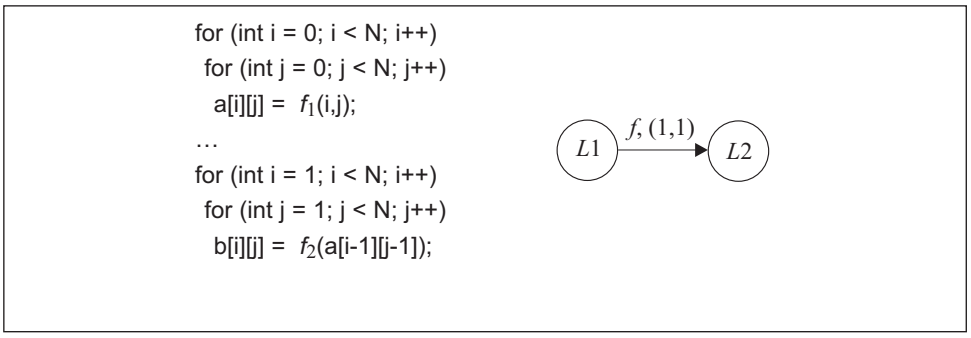



*Рис. 2*

Важность преобразования циклов в исходном тексте программ была замечена достаточно давно [7]. Целью большинства работ, посвященных методам преобразования циклов, было улучшение распараллеливания вычислений. Эффективность распараллеливания, однако, лишь частично связана с локальностью доступа к данным, под которой подразумевается повторное использование одних и тех же данных по прошествии относительно небольшого промежутка времени [8]. Во многих работах преобразования циклов проведены с использованием графов [5, 6, 8].

**Объединение многомерных циклов.** Основным источником энергопотребления схем памяти являются операции чтения и записи. Как видно из рис. 1, операция объединения циклов позволяет преобразовать исходный текст приложения так, чтобы сократить число обращений к ОЗУ.

Предлагаемый способ позволяет объединять многомерные циклы одинаковой размерности, между которыми существуют связи по данным и выходу. Поскольку ограничения по объему статьи не позволяют привести алгоритм объединения циклов полностью, рассмотрим только основные идеи, реализованные в алгоритме, и примеры их применения: построение графов, соответствующих исходному тексту приложения; объединение циклов со связью по данным; объединение циклов со связью по выходу; минимизация количества данных, которые необходимо хранить в кэш памяти или регистрах процессора. Для простоты изложения будем рассматривать только примеры двумерных циклов.

***Построение графа вычислений, соответствующего исходному тексту приложения.*** На рис. 2 представлены исходный текст программы, содержащий два цикла, обрабатывающих двумерные массивы a и b, и соответствующий граф вычислений. Вершины $L1$ и $L2$ графа соответствуют вычислению элементов массивов a и b. Между циклами существует зависимость по данным, которая представлена в графе ребром, направленным от вершины $L1$ к вершине $L2$. Ярлык $f$ используется для обозна-

чения типа зависимости по данным, в отличие от типа связи по выходу. Ребру присваивается набор весов, число которых равно размерности массивов. Величины весов вычисляются так.

Допустим, элементы массива b вычисляются с помощью элементов массива $a[i_1\text{-}d_1, i_2\text{-}d_2, \ldots, i_n\text{-}d_n]$. Вес, соответствующий $i_k$ итерационной переменной, вычисляется как разница $k$-х значений индексных выражений для элементов массива a при вычислении массивов a и b: $i_k - (i_k - d_k) = d_k$. Тогда набор весов данного ребра будет таким: ($d_1$, $d_2$, …, $d_n$).

Рассмотрим случай, когда циклы не могут быть объединены. На рис. 3 представлены исходный текст — циклы которые невозможно объединить, и граф вычислений, соответствующий наличию отрицательных значений весов ребра, соединяющего вершины $L1$ и $L2$.

***Объединение циклов со связью по данным.*** На рис. 4, 5 и 6 показано объединение многомерных циклов со связью по данным. В соответствии с методом объединения одномерных циклов [9] для объединения многомерных циклов, необходимо трансформировать граф вычислений таким образом, чтобы в нем не осталось $f$-ребер с отрицательными значениями весов.

Перед началом преобразования графа присвоим каждой его вершине набор весов, число которых равно размерности массивов. В исходном тексте (рис. 4, *а*) все значения весов в наборе нулевые.

Для вершин и $f$-ребер исходного графа (рис. 4, *б*) изменение каждого веса в наборе, соответствующего одной из итерационных переменных, выполняется так же, как и в одномерном случае. Например, на рис. 5 вес ребра $L2 \rightarrow L3$, соответствующий итерационной переменной $j$, был увеличен на единицу и стал равен нулю. Как следствие вес ребра $L2 \rightarrow L4$, соответствующий итерационной переменной $j$, увеличился на единицу и стал равен единице. Вес ребра $L1 \rightarrow L2$, соответствующий итерационной переменной $j$, уменьшился на единицу и стал равен –1. Вес вершины $L2$, соответствующий итерационной переменной $j$, уменьшился на единицу и стал равен –1.

Таким образом, если вес какого-либо ребра, соответствующий $k$-й итерационной переменной, увеличивается на определенную величину, то соответствующий $k$-й итерационной переменной вес вершины, из которой выходит данное ребро, уменьшается на данную величину. Вес, соответствующий $k$-й итерационной переменной ребер, входящих в эту вершину, также уменьшается на ту же величину, а вес, соответствующий $k$-й итерационной переменной ребер, выходящих из этой вершины, увеличивается на упомянутую величину.

```
for (int i = 0; i < N; i++)
 for (int j = 0; j < N; j++)
  a[i][j] = f1(i,j);
…
for (int i = 0; i < N - 1; i++)
 for (int j = 0; j < N - 1; j++)
  b[i][j] = f2(a[i+1][j+1]);
```

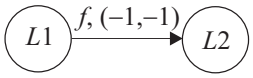
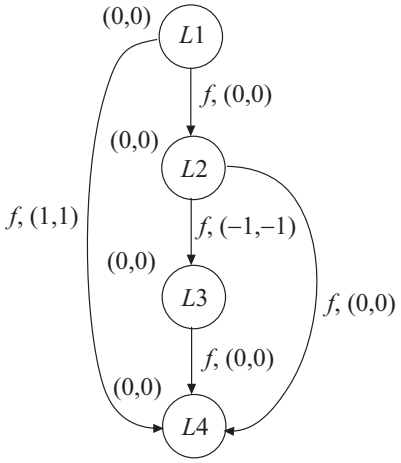



*Рис. 3*

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

*а*




*б*

*Рис. 4*

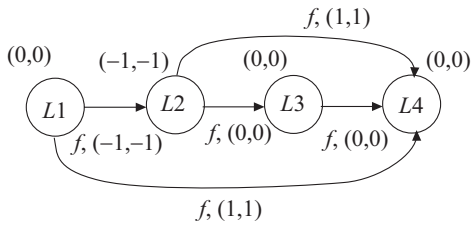



*Рис. 5*

```
for (i = 2; i < N; i++)
 for (i = 2; j < N; j++) {
  a1[i][j] = f1(i,j);
  a2[i][j] = f2(a1[i][j]);
  a3[i-1][j-1] = f3(a2[i][j]);
  a4[i-1][j-1] =
f4(a1[i-2][j-2],a2[i-1][j-1],a3[i-1][j-1]);
}
```

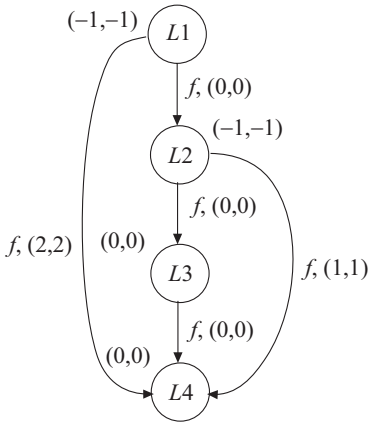



*Рис. 6*

```
for (int i = 0; i < N; i++)
 for (int j = 0; j < N; j++)
      a[i][j] = f1(i,j);
…
for (int i = 1; i < N; i++)
 for (int j = 1; j < N; j++)
  b[i][j] = f2(a[i][j],a[i-1][j-1]);
…
for (int i = 0; i < N - 1; i++)
 for (int j = 0; j < N - 1; j++)
 a[i+1][j+1] = f3(b[i][j]);
```

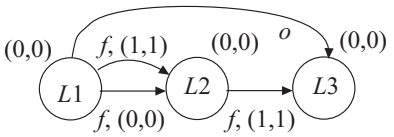
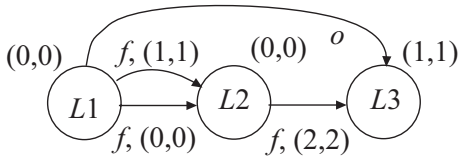



*Рис. 7*

Двигаясь к вершине $L1$ (началу графа), проводим аналогичные преобразования для всех весов в наборах. На рис. 6 показаны текст объединенного цикла и конечный преобразованный граф. Вес вершин графа используется при формировании текста объединенного цикла.

***Объединение циклов со связью по выходу.*** На рис. 7 представлены исходный текст программы, содержащий три цикла, и граф, соответствующий данному тексту. Каждой вершине присваивается набор нулевых весов. Веса $f$-ребер вычисляются так же, как это сделано выше. Ребро с ярлыком $o$ отображает наличие связи по выходу и не имеет веса. Между первым и третьим циклами существует связь по выходу.




```
for (int i = 2; i < N; i++)
 for (int j = 2; j < N; j++)
{
   a[i][j] = f1(i,j);
   b[i][j] = f2(a[i][j],a[i-1][j-1]);
   a[i-1][j-1] = f3(b[i-2][j-2]);
}
```

*Рис. 8*

В исходном виде данные три цикла объединить невозможно. Например, значение элемента a[1,1] необходимо для вычисления b[2,2], но оно уже перезаписано во время выполнения третьего цикла.

Преобразованный граф для данного случая и текст объединенного цикла показаны на рис. 8.

Объединение циклов возможно в случае перезаписи значения a[i,j] после его использования при вычислении b[i+1,j+1]. Для этого трансформируем граф следующим образом. Увеличиваем каждый $k$-й вес вершины $L3$ (на которой заканчивается $o$-ребро) на величину, равную максимальному значению из всех $k$-х весов всех $f$-ребер, исходящих из вершины $L1$ (из которой исходит $o$-ребро).

Например, из вершины $L1$ исходят два $f$-ребра. Первый вес из набора весов одного из них равен единице, а другой — нулю. Максимальное значение одной из двух указанных вершин равно единице, поэтому первый из набора весов вершины $L3$ должен увеличиться на единицу. При этом веса всех $f$-ребер, входящих в $L3$, должны увеличиться на ту же величину, на которую увеличились соответствующие веса вершины $L3$, а веса всех исходящих из данной вершины $f$-ребер должны уменьшиться на упомянутую величину.

***Минимизация количества данных, которые необходимо хранить в кэш памяти или регистрах процессора.*** Назовем кэш память и регистры процессора быстрой памятью. На рис. 9, *а* представлен исходный текст программы, содержащий три цикла, которые можно объединить в один, ничего не меняя в записи вычислений (рис. 9, *б*). Для того чтобы уменьшить число операций чтения из ОЗУ, необходимо хранить в быстрой памяти значения четырех элементов массива a (a[i,i-1][j,j-1]) и девяти элементов массива b (b[i,..,i-2][j,..,j-2]).

Исходные циклы можно объединить и так, чтобы хранить в быстрой памяти минимум данных, а именно только два значения элементов мас-

```
…
for (int i = 0; i < N; i++)
 for (int j = 0; j < N; j++)
  a[i][j] = i+j+1;
…
for (int i = 1; i < N; i++)
 for (int j = 1; j < N; j++)
  b[i][j] = 2*a[i-1][j-1]+1;
…
for (int i = 2; i < N; i++)
 for (int j = 2; j < N; j++)
  c[i][j] = b[i-2][j-2]+3;
…
  c[i][j] = b[i-2][j-2]+3;
```

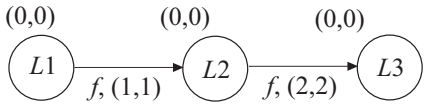
*а*

```
…
for (int i = 2; i < N; i++)
 for (int j = 2; j < N; j++) {
  a[i][j] = i+j+1;
  b[i][j] = 2*a[i-1][j-1]+1;
  c[i][j] = b[i-2][j-2]+3;
}
…
```

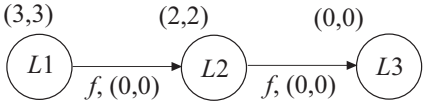
*б*

```
…
for (int i = 0; i < N - 3; i++)
 for (int j = 0; j < N - 3; j++) {
  a[i+1][j+1] = [i][j]+1;
  b[i+1][j+1] = 2*a[i][j]+1;
  c[i+3][i+3] = b[i+1][i+1]+3;
}
…
```

*в*




*г*




*д*

*Рис. 9*

сивов: a[i][j] и b[i+1][j+1] (рис. 9, *в*). Данный текст цикла получается с помощью преобразований исходного графа, представленного на рис. 9, *г*.

При минимизации веса всех *f*-ребер графа вес каждого ребра должен оставаться неотрицательным. Начиная с вершины $L3$ и двигаясь к вершине $L1$, преобразуем веса всех *f*-ребер. Уменьшаем на два, т.е. до нуля, вес ребра $L2 \to L3$. При этом вес исходной для данного ребра вершины $L2$ увеличивается на два, вес ребра $L1 \to L2$ также увеличивается на два и становится равен трем. Затем уменьшаем на три, т.е. до нуля, вес ребра $L1 \to L2$, увеличивая одновременно на три вес вершины $L1$.

Преобразованный граф представлен на рис. 9, *д*. Все *f*-ребра данного графа имеют одинаковый нулевой вес. Это соответствует тому, что при вычислении значения элемента массива b необходимо хранить в быстрой памяти только одно значение элемента массива a, а при вычислении зна-

```
…
for ( i=0; i<50; i++) a1[i] = i;
for (i = 0; i<50; i++) a2[i] = a1[i] * 2;
for (i = 0; i<50; i++) a3[i] = a2[i] - 1;
for (i = 0; i<50; i++) a4[i] = a3[i] * 2 + 1;
…
```

*а*

```
…
for (i = 0; i < 50; i++) {
   a1 = i;
   a2 = a1 * 2;
   a3 = a2 - 1;
   a4[i] = a3 * 2 + 1;
}
…
```

*б*

*Рис. 10*

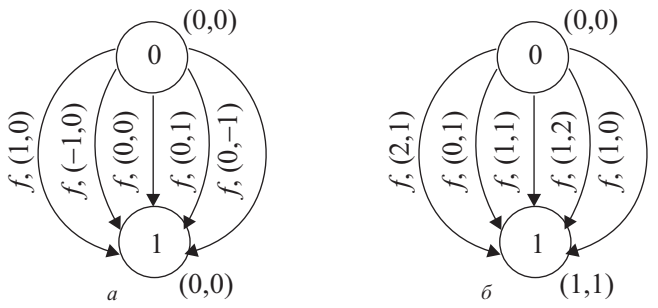



*Рис. 11*

чения элемента массива c необходимо хранить в быстрой памяти только одно значение элемента массива b (см. 9, *в*).

***Результат эксперимента.*** Уменьшение энергопотребления цифровой системы после объединения циклов в исходном тексте программы проверено экспериментально с использованием микроконтроллера MSP430F435 фирмы Texas Instruments. Рассмотрен исходный текст приложения, представленный на рис. 10, *а*. С помощью алгоритма объединения циклов исходный текст преобразован в текст, приведенный на рис. 10, *б*.

Время выполнения программ с помощью указанного микроконтроллера одинаково. В случае выполнения исходной программы ток потребления микроконтроллера составлял 680 мкА, для программы с объединенным циклом — 538 мкА. В результате операции объединения циклов в тексте программы ток потребления микроконтроллера удалось уменьшить приблизительно на 21 %.

**Пример преобразования текста приложения.** Предлагаемый алгоритм объединения многомерных циклов использован для преобразования текста программы двух операций фильтрации исходного изображения:

1) с помощью прямоугольного сглаживающего фильтра радиуса 1;

2) с помощью контрастоповышающего фильтра с ядром, задаваемым матрицей

$$M_{1\text{contr}} = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}.$$

В исходном тексте программы обработка изображения выполняется в двух двумерных циклах:

```
…
for (x=0; x<N; ++x)
  for (y=0; y<M; ++y)
    image_im[x][y] = (image_in[x-1][y-1]+ image_in[x-1][y]+
    image_in[x-1][y+1]+image_in[x][y-1]+image_in[x][y]+image_in[x][y+1]+
    image_in[x+1][y-1]+image_in[x+1][y]+image_in[x+1][y+1])/9;
        for (x=0; x<N; ++x)
  for (y=0; y<M; ++y)
    image_out[x][y] = 5*image_im[x][y]-image_im[x-1][y]-image_im[x+1][y]-
    image_im[x][y-1]-image_im[x][y+1];
…
```

В соответствии с исходным текстом программы строим исходный граф вычислений (рис. 11, *а*), который состоит из четырех вершин и *f*-ребер, отображающих использование элементов одних массивов при вычислении других. Вершина с номером 0 отображает вычисление массива

| $N \times M$ | Число операций чтения | | Уменьшение числа операций чтения, % |
|---|---|---|---|
| | для исходной программы | для программы с объединенным циклом | |
| 160× 120 | 76 800 | 21 342 | 72,2 |
| 320× 240 | 307 200 | 81 102 | 73,6 |
| 640× 400 | 1024 000 | 264 142 | 74,2 |
| 720× 576 | 1 658 880 | 424 638 | 74,4 |
| 800× 600 | 1 920 000 | 490 782 | 74,4 |
| 1024× 768 | 3 145 728 | 800 238 | 74,6 |
| 1280× 800 | 4 096 000 | 1 040 302 | 74,6 |
| 1440× 900 | 5 184 000 | 1 314 342 | 74,6 |
| 1680× 1050 | 7 056 000 | 1 785 402 | 74,7 |
| 1600× 1200 | 7 680 000 | 1 941 582 | 74,7 |
| 1920× 1200 | 9 216 000 | 2 328 462 | 74,7 |

«image_im». Вершина с номером 1 — вычисление массива «image_out». Поскольку упомянутые массивы двумерные, все вершины и *f*-ребра обладают набором из двух весов. Первый вес соответствует итерационной переменной *x*, второй — итерационной переменной *y*. После соответствующих преобразований получаем итоговый граф (рис. 11, *б*) и следующий текст объединенного цикла:

```
…
for (x=1; x<N; ++x) {
 for (y=1; y<M; ++y) {
   image_im[x][y] = (image_in[x-1][y-1]+image_in[x-1][y]+
   image_in[x-1][y+1]+image_in[x][y-1]+image_in[x][y]+image_in[x][y+1]+
   image_in[x+1][y-1]+image_in[x+1][y]+image_in[x+1][y+1])/9;
   image_out[x-1][y-1] = 5*image_im[x-1][y-1]-image_im[x-2][y-1]-
   image_im[x][y-1]-image_im[x-1][y-2]-image_im[x-1][y];
 }
}
…
```

При выполнении программы в исходном виде выполняется $2\times N\times M$ операций чтения из ОЗУ и $2\times N\times M$ операций записи в ОЗУ. При выполнении преобразованной программы выполняется столько же операций записи в ОЗУ, но на $3\times(N-2)\times(M-3)$ меньше операций чтения из ОЗУ вследствие хранения вычисленных данных в кэш памяти в процессе вычисления массивов в объединенном цикле.

В таблице представлены результаты расчета выигрыша в числе операций чтения из ОЗУ для стандартных значений размеров графических изображений. Как видно из данных, приведенных в таблице, такой выигрыш для большинства стандартных размеров изображений составляет приблизительно 75 %.

**Выводы.** Предлагаемый способ объединения многомерных циклов, обрабатывающих массивы одинаковой размерности, в исходном тексте описания цифровых систем дает возможность уменьшить число обращений к ОЗУ, что приводит к уменьшению энергопотребления цифровой системы.

Данный метод объединения циклов более эффективен, чем предлагаемый в [6], так как позволяет объединять циклы со связью по выходу и анализировать влияние связей внутри одного цикла на возможность объединения его с другим.

Предлагаемый способ объединения циклов является основой для автоматизации трансформации текста программ на языках высокого уровня.

A method of multidimensional loop fusion is presented in this paper. This method of code-to-code transformations serves for power consumption reduction of digital systems through reduction of the number of accesses to the main memory. The algorithm is a basis for automation of code-to-code transformations.

1. *Rabaey J.* Low Power Design Essentials — Dordrecht: Springer, 2009. — 288 p.
2. *Verma M., Marwedel P.* Advanced memory optimization techniques for low-power embedded processors. — Dordrecht: Springer, 2007. — 161 p.
3. *Sproch J.* High Level Power Analysis and Optimization// Tutorial. International Symposium on Low Power Electronics and Design. — N Y : ACM Press, 1997. — 10 p.
4. *International* Technology Roadmap for Semiconductors — http://public.itrs.net/
5. *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.
6. *Fraboulet A., Kodary K., Mignotte A.* Loop fusion for memory space optimization// The 14th International Symposium on System Synthesis. Piscataway. — Ibid. — 2001. — P. 95—100.
7. *Bacon D., Graham S., Sharp O.* Compiler transformations for high-performance computing//ACM Computing Surveys. — 1994. — **26**, № 4. — P. 345 — 420.
8. *Ахо А. В., Лам М. С., Ульман Д. Д.* Компиляторы: принципы, технологии и инструментарий. — М. : Вильямс. — 2008. — 1184 с.
9. *Лазоренко Д. И.* Алгоритм объединения одномерных циклов исходного текста описания цифровых систем с целью снижения их энергопотребления.// Сб. тр. «Системи обробки інформації». — Вип. 8(66). Харківський університет повітряних сил ім. Івана Кожедуба. — 2007. — С. 45 — 49.

Поступила 17.06.10;
после доработки 19.07.10

*ЛАЗОРЕНКО Дмитрий Иванович, канд. техн. наук, мл. науч. сотр. 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 путем высокоуровневых трансформаций исходного описания приложения на языках высокого уровня C/C++, SystemC, VHDL.*

*ЧЕМЕРИС Александр Анатольевич, канд. техн. наук, ст. науч. сотр., ученый секретарь Ин-та проблем моделирования в энергетике им. Г. Е. Пухова НАН Украины. В 1982 г. окончил Киевский политехнический ин-т. Область научных исследований — параллельные компьютеры, специализированные архитектуры, системы автоматического проектирования.*

*ТАРАПАТА Валерий Владимирович, мл. науч. сотр. Ин-та проблем моделирования в энергетике им. Г. Е. Пухова НАН Украины. В 2003 г. окончил Национальный технический университет Украины «Киевский политехнический ин-т». Область научных исследований — разработка и исследование голографических систем эхоскопии.*