УДК 621.3

Лазоренко Д.И.

АЛГОРИТМ ОБЪЕДИНЕНИЯ МНОГОМЕРНЫХ ЦИКЛОВ ИСХОДНОГО ТЕКСТА ОПИСАНИЯ ЦИФРОВЫХ СИСТЕМ С ЦЕЛЬЮ СНИЖЕНИЯ ИХ ЭНЕРГОПОТРЕБЛЕНИЯ

Институт проблем моделирования в энергетике НАН Украины

Предложен метод объединения многомерных циклов на уровне исходного кода с целью понижения энергопотребления проектируемых устройств за счет уменьшения обращений к основной памяти. Алгоритм является основой для автоматизации трансформации кода программ на языках высокого уровня.

Введение

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

Современные цифровые системы, например, мультимедийные приложения, переносные телефоны, карманные персональные компьютеры, обрабатывают большие массивы данных по сложным алгоритмам. 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, проявления эффекта электромиграции и электромагнитного излучения.

Источники энергопотребления в КМОП схемах

Подавляющее число современных микросхем производится с помощью КМОП (Комплиментарный Металл-Оксид-Диэлектрик) 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.

$$P_{total} = P_{short} + P_{stat} + P_{leak} + P_{dyn}.$$

Токи короткого замыкания. Токи короткого замыкания существуют в процессе нормального функционирования логических схем. Под ними понимаются токи, протекающие через транзисторы из шины питания непосредственно в шину земли.

Статические токи. Под статическими понимаются временные или постоянные токи.

Паразитные токи утечки. Под паразитными токами утечки понимаются подпороговые токи транзисторов и токи в подложке.

Динамическое рассеяние энергии. Динамическое рассеяние энергии происходит из-за зарядки/разрядки узлов схемы, и может быть представлено следующей формулой:

$$P_{dyn} = C \cdot V_{dd}^{\ 2} \cdot \alpha \cdot f \ ,$$

где C – суммарная ёмкость в узлах схемы, V_{dd} – величина напряжения питания, f – частота переключений, α - коэффициент переключательной активности (среднее число логических вентилей, переключающихся в течение одного цикла сигнала синхронизации) [1].

Динамическое рассеяние энергии может составлять до 80% от полных потерь энергии. Переключательная активность в значительной мере определяется программным обеспечением цифровой системы [2, 3].

Потенциальный выигрыш в энергопотреблении

В работе [4] обоснован вывод, что потенциальный выигрыш в энергопотреблении тем выше, чем выше уровень абстракции процесса проектирования, на котором принимается решение. Для системного уровня выигрыш может быть 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 схемам памяти присущи большие паразитные токи утечки. Кроме того, на обращение к памяти тратится много энергии, например, на операцию чтения из внешней памяти расходуется в 33 раза более энергии, чем на операцию 16-битного сложения. Согласно прогнозу международной организации *International Technology Roadmap for Semiconductors* схемы памяти будут занимать всё больше площади на полупроводниковых кристаллах: в 2008 г. – 83%, в 2011 г. – 90%, в 2014 г. – 94%. Также, величина динамического энергопотребления схем памяти в ближайшем будущем будет только увеличиваться. Например, по прогнозу на 2006 г. динамическое рассеяние энергии на схемах памяти будет на 25% больше, чем на логических схемах, к 2010 г. эта величина увеличиться до 2 раз, к 2015г. – до 2,5 раз, к 2020 г. – до 3 раза [5].

Очевидно, в процессе проектирования нужно добиваться уменьшения объёма требуемой приложению памяти и количества обращений к ней. Для этого необходимо 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 объёмы данных по сложным алгоритмам [6 – 8].

Для снижения объёма требуемой памяти необходимо уменьшить размер и количество временных массивов, создаваемых в процессе обработки данных. Уменьшения объёма памяти можно также добиться путём повторного использования одних и тех же адресов памяти разными массивами.

Алгоритм объединения многомерных циклов

Известно, что основным источником энергопотребления схем памяти являются операции чтения и записи. В данной статье предлагается алгоритм анализа циклов на возможность их объединения, что позволяет уменьшить количество обращений к памяти и ее объем. Данный алгоритм использует представление исходного текста программы в виде графов и основан на предложенном в [9] графическом методе объединения одномерных циклов исходного текста описания цифровых систем. Рассмотрим далее формирование и преобразование упомянутых графов.

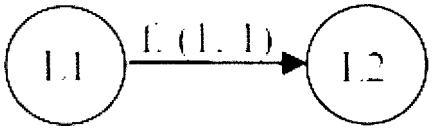
Для наглядности на рисунках будут использоваться двумерные массивы. Пусть есть некоторый исходный текст программы, содержащий многомерные циклы (рис. 1а). На рис. 1б изображён граф исходного текста программы. Его вершины *L1* и *L2* соответствуют вычислению элементов массивов ***a*** и ***b***. Между циклами существует зависимость по данным, эта зависимость представлена в графе дугой направленной от вершины *L1* к вершине *L2*. Дуге присваивается набор весов, количество которых равно размерности массивов. Сами величины весов вычисляются следующим образом. Допустим, элементы массива $b[i_1,i_2,\ldots,i_n]$ вычисляются при помощи элементов массива $a[i_1-d_1,i_2-d_2,\ldots,i_n-d_n]$. Вес, соответствующий i_k итерационной переменной, будет вычисляться, как разница k-х значений индексов для элементов массива ***b*** и ***a***: $i_k-(i_k-d_k)=d_k$. Тогда набор весов данной дуги будет таким - $(d_1,d_2,\ldots,d_n)$. Ярлык *«f»* нужен для обозначения типа зависимости по данным, поскольку ниже будет рассматриваться также тип связи по выходу.

Пусть элементы массива вычисляются внутри цикла, используя значения других элементов этого же массива (рис. 2а). В графе данная ситуация отображается наличием направленных дуг, которые начинаются и заканчиваются на одной и той же соответствующей определённому массиву вершине графа (рис. 2б).

Дуге присваивается набор весов, количество которых равно размерности массивов. Пусть элементы массива $a[i_1,i_2,\ldots,i_n]$ вычисляются при помощи других элементов этого же массива $a[i_1-d_1,i_2-d_2,\ldots,i_n-d_n]$, тогда вес, соответствующий i_k итерационной переменной, будет вычисляться как разница k-х значений индексов для элементов массива **a**: $i_k-(i_k-d_k)=d_k$. Циклы, приведенные на рис. 2а, невозможно объединить. Этому на графе соответствует наличие у соединённых f-дугой вершин *L1* и *L2* замкнутых направленных дуг с наборами весов

```
...
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-1][j-1]);
...
```

а) исходный текст

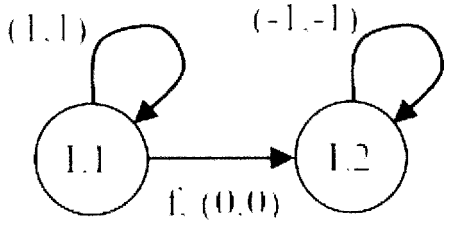


б) граф вычислений

Рис. 1. Текст программы и соответствующий ему граф

```
...
for (int i = 1; i < N; i++)
 for (int j = 1; j < N; j++)
  a[i][j] = f1(a[i-1][j-1]);
...
for (int i = N - 2; i => 0; i--)
 for (int j = N - 2; j => 0; j--)
  b[i][j] = f2(b[i+1][j+1], a[i][j]);
...
```

а) исходный текст программы



б) граф вычислений

Рис. 2. Текст программы и соответствующий ему граф

такими, что k-е веса имеют противоположные знаки.

На рис. 3а представлен некоторый исходный текст. Циклы в исходном виде невозможно объединить. Данной ситуации соответствует наличие отрицательных значений весов дуги, соединяющей вершины *L1* и *L2*, на рис. 3б.

В работе [9] описан способ трансформации графа представления зависимостей при вычислении одномерных циклов, благодаря которому становится возможным их объединение. Ниже предлагается основанный на упомянутом способе алгоритм преобразования графов для многомерных циклов.

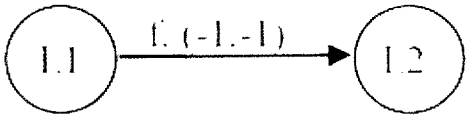
Перед началом преобразования графа присвоим каждой его вершине набор весов, количество которых равно размерности массивов. В исходном состоянии все значения весов в наборе будут нулевыми. Данные значения весов будет изменять по правилам приводимым ниже.

Рассмотрим преобразования графа на примере некоторого исходного текста программы на рис. 4а. На рис. 4в показан исходный граф.

Аналогично тому, как было сделано в работе [9], необходимо трансформировать граф таким образом, чтобы в нём не осталось f-дуг с отрицательными значениями

```
...
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]);
...
```

а) исходный текст программы

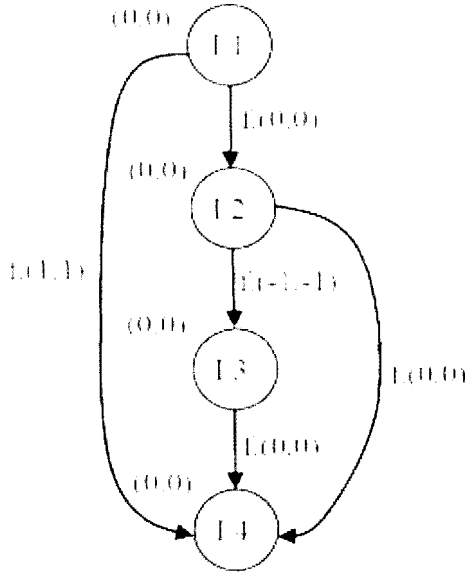


б) граф вычислений

Рис. 3. Текст программы и соответствующий ему граф

```
...
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) исходный текст программы



б) граф вычислений

Рис. 4. Текст программы и соответствующий ему граф

весов. Для вершин и *f*-дуг изменение каждого веса в наборе, соответствующего одной из итерационных переменных, производится точно так же, как и в одномерном случае. Например, как видно на рис. 5 вес, соответствующий итерационной переменной *j*, дуги *L2→L3* был увеличен на 1 и стал равным 0. Как следствие, вес, соответствующий итерационной переменной *j*, дуги *L2→L4* увеличился на 1 и стал равным 1. Вес, соответствующий итерационной переменной *j*, дуги *L1→L2* уменьшился на 1 и стал равен - 1. Вес, соответствующий итерационной переменной *j*, вершины *L2* уменьшился на 1 и стал равен - 1. Таким образом, если вес, соответствующий *k*-ой итерационной переменной, какой-либо дуги увеличивается на определённую величину, то вес, соответствующий *k*-ой итерационной переменной, вершины, из которой выходит данная дуга, уменьшается на данную величину. Вес, соответствующий *k*-ой итерационной переменной, дуг, входящих в эту вершину, также уменьшается на ту же величину, а вес, соответствующий *k*-ой итерационной переменной, дуг, выходящих из этой вершины, увеличивается на упомянутую величину.

Затем, двигаясь к вершине *L1* (началу графа), проводим аналогичные преобразования для всех весов в наборах. На рис. 6 показан конечный преобразованный граф и текст объединённого цикла.

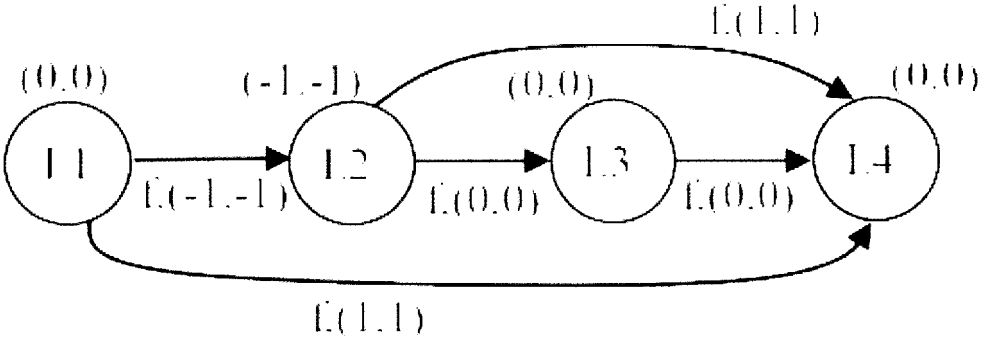
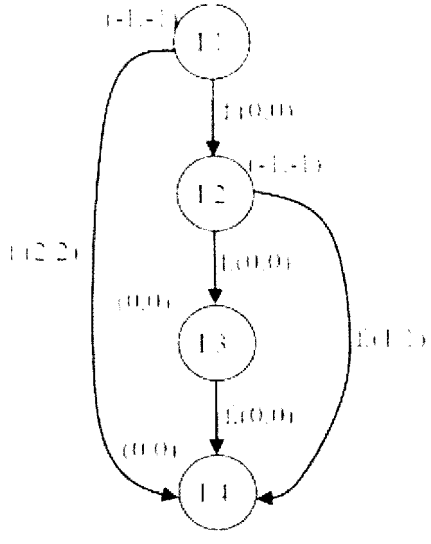


Рис. 5. Граф вычислений в процессе преобразований

```
...;
for (i = 2; i < N; i++)
 for (j = 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]);
}
...;
```

а) текст объединенного цикла



б) итоговый граф

Рис. 6. Текст программы и соответствующий ему граф

Вес вершин графа используется при формировании текста объединенного цикла.

Теперь рассмотрим объединение циклов, если между ними присутствует связь по выходу. На рис. 7а представлен некоторый исходный текст программы, содержащий три цикла. На рис. 7б представлен граф, соответствующие исходному тексту программы на рис. 7а. Каждой вершине присваивается набор нулевых весов. Веса *f*-дуг вычисляются так же, как это было сделано выше. Дуга с ярлыком «*o*» отображает наличие связи по выходу и не имеет веса. Между первым и третьим циклами существует связь по выходу.

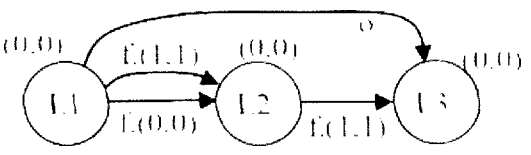
В исходном виде данные три цикла объединить невозможно. Например, значение элемента *a* [1,1] необходимо для вычисления *b* [2,2], но до этого оно уже перезаписывается во время выполнения третьего цикла.

Итоговый граф для данного случая показан на рис. 8а, а текст объединенного цикла – на рис. 8б.

Чтобы сделать объединение циклов возможным, необходимо перезаписывать значение *a[i,j]* уже после того, как оно было использовано для вычисления *b[i+1,j+1]*. Для этого требуется трансформировать граф следующим образом. Увеличиваем каждый *k*-ый вес вершины *L3* (на которой заканчивается *o*-дуга) на величину, равную максимальному значению из всех *k*-ых весов всех *f*-дуг, исходящих из вершины *L1* (из которой исходит та же *o*-дуга). Например, из вершины *L1* исходят две *f*-дуги, первый вес из набора одной из них равен 1, для другой – эта величина равна 0. Максимальное значение из приведенных двух равно 1, поэтому первый из набора весов вершины *L3* должен увеличиться на 1. При этом веса всех *f*-дуг, входящих в *L3*, должны

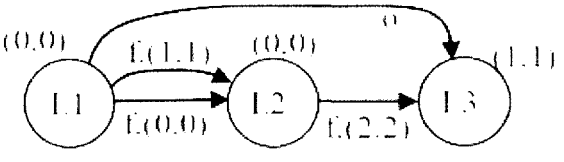
```
...
for (int i = 0; i < N; i++)
 for (int j = 0; j < N; j++)
  a[i][j] = f1(i,j);
...
for (int i = 1; i < N; i++)
 for (int j = 1; j < N; j++)
  b[i][j] = f2(a[i][j],a[i-1][j-1]);
...
for (int i = 0; i < N - 1; i++)
 for (int j = 0; j < N - 1; j++)
  a[i+1][j+1] = f3(b[i][j]);
...
```

а) исходный текст программы



б) исходный граф

Рис. 7. Текст программы и соответствующий ему граф



а) итоговый граф

```
...
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]);
}
...
```

а) итоговый граф

Рис. 8. Преобразованный граф и соответствующий ему текст программы

увеличиться на ту же величину, на которую увеличились соответствующие веса вершины *L3*, веса же всех исходящих из данной вершины *f*-дуг должны уменьшиться на упомянутую величину.

Для обобщения, ниже приводятся алгоритмы построения и преобразования графического отображения циклов.

Алгоритм построения графа исходного текста программы:

1. Каждому вычисляемому в циклах массиву ставится в соответствие вершина графа с начальным набором нулевых весов.

2. Вершинами соединяются направленными дугами. Дуги исходят из вершин, соответствующих тем массивам, которые вычисляются первыми в исходном тексте программы.

2.1. Дуги, соответствующие связи по выходу, не имеют веса и обозначаются ярлыком *«o»*.

2.2. Дуги, соответствующие связи по данным, обозначаются ярлыком *«f»*. Каждой *f*-дуге ставится в соответствие набор весов. Размерность набора равна размерности массивов. *K*-ый вес в наборе соответствует *k*-ой итерационной переменной. Начальное значение *k*-го веса в наборе определяется как разница между индексными выражениями для *k*-ой индексной переменной массива, который вычисляется в левой части выражения, соответствующего вершине, из которой выходит дуга, и правой части выражения, использующего этот массив и соответствующего вершине, на которой дуга заканчивается.

2.2.1. Проиллюстрируем пункт 2.2 на примере рис. 1. Между первым и вторым циклами существует связь по данным (массив ***a***), поэтому из вершины *L1*, соответствующей вычислению массива ***a***, выходит дуга в вершину *L2*, соответствующую вычислению массива ***b***. Дуга отмечается ярлыком *«f»*. Индексное выражение для массива ***a*** для индексной переменной i в левой части выражения, соответствующего вершине *L1*, есть *i*. В правой части выражения, соответствующего вершине *L2* – *i-1*. Разница этих индексных выражений есть *i–(i-1)=1*. Проделав те же вычисления для индексной переменной *j*, получаем 1. Поэтому начальные значения весов в наборе для дуги будут – (1,1).

Алгоритм преобразования графа выглядит следующим образом:

1. Если существуют *«f»*-дуги с отрицательными значениями весов в наборе, тогда, двигаясь от такой дуги к началу

графа, сделать все значения весов «*f*»-дуг неотрицательными следующим образом: пусть *k*-ый вес имеет отрицательное значение, увеличить *k*-ый вес дуги до нуля на необходимую величину, затем уменьшить *k*-ый вес вершины, из которой вышла данная дуга, на упомянутую величину; все другие «*f*»-дуги, исходящие из данной вершины, увеличивают свой *k*-ый вес на указанную величину, а «*f*»-дуги входящие в эту вершину, наоборот, на данную величину свой *k*-ый вес уменьшают.

2. Если никакими преобразованиями нельзя сделать значения весов в наборе «*f*»-дуг неотрицательными, то все исследуемые циклы объединить невозможно.

3. В случае, если все циклы не могут быть объединены, разбить циклы на группы, которые расположены выше и ниже вершин, соединенных «*f*»-дугами с отрицательными значениями весов в наборах.

3.1. Если внутри какой-либо из групп есть циклы со связями по выходу, то *k*-ый вес вершины, в которую входит «*o*»-дуга, должен быть увеличен на величину, равную максимальному значению из всех *k*-х весов «*f*»-дуг, выходящих из вершины исходной для данной «*o*»-дуги. При этом веса всех *f*-дуг, входящих в вершину, в которую входит «*o*»-дуга, должны увеличиться на ту же величину, на которую увеличились соответствующие веса этой вершины, веса же всех исходящих из данной вершины *f*-дуг должны уменьшиться на упомянутую величину.

Выводы

Предлагаемый алгоритм представляет собой способ анализа циклов с точки зрения возможности их объединения. Данный способ позволяет добиться лучших результатов в объединении циклов, чем, например, предлагаемый в [7]. Также, данный способ позволяет анализировать влияние связей внутри одного цикла на возможность объединения его с другим, что отсутствует в методе, предлагаемом в [7]. В результате объединения циклов, выполненного согласно предлагаемому методу, можно существенно сократить количество операций обращения к медленной памяти, и уменьшить ее размер. Предлагаемый алгоритм позволяет также минимизировать количество данных, которые необходимо хранить в быстрой памяти в процессе вычислений.

Автор планирует рассмотреть вопросы автоматизации данного метода для ввода его в процесс оптимизации кода компилятором с языка *SystemC*.

Список литературы

1. *Veendrick H.J.M.* Deep-Submicron CMOS ICs. From Basics to ASICs. - Kluwer academic publishers, 2000. – 539 p.

2. *Poppen F.* Low Power Design Guide. – OFFIS Research Institute. http://www.offis.de.

3. *Kim H.S., Irwin M.J., Vijaykrishnan N., Kandemir M.* Effect of compiler optimizations on memory energy // 2000 IEEE Workshop on Signal Processing Systems. SiPS 2000. – 2000. – P. 663 – 672.

4. *Sproch J.* High Level Power Analysis and Optimization. Tutorial // 1997 International Symposium on Low Power Electronics and Design. – 1997.

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. Proceedings (ISSS'99). 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. Proceedings 2001. – 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. – 1994. – P. 178 – 187.

9. *Лазоренко Д.И.* Алгоритм объединения одномерных циклов исходного текста описания цифровых систем с целью снижения их энергопотребления.//Системи обробки інформації. - Х.:ХУПС, 2007. – Вип. 8(66). – С. 2–12.