УДК 621.3

Д.И. Лазоренко, канд. техн. наук

Объединение многомерных циклов и его применение

Рассмотрен способ объединения многомерных циклов и его применение для уменьшения времени выполнения вычислений и снижения энергопотребления встроенных систем.

A method of uniting of multidimensional loops and its application for low power embedded systems and high-performance computing.

Ключевые слова: *объединение циклов, встроенные системы*

Введение

Современное программное обеспечение использует большие объемы памяти. Схемы памяти могут занимать от 50% до 80% площади полупроводникового кристалла. Известно, что схемам памяти присущи большие паразитные токи утечки. С переходом на каждое новое поколение 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 mV 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 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 2011 г. – 90%, в 2014 г. – 94%. Также, величина динамического энергопотребления схем памяти в ближайшем будущем будет только 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 увеличиться до 2,5 раз, к 2020 г. – до 3 раза [1]. Согласно тому же прогнозу, потери энергии на схемах памяти за счёт паразитных токов утечки будут только расти. Таким образом, в процессе проектирования нужно добиваться уменьшения объёма требуемой приложению памяти и количества обращений к ней.

Важность оптимизации работы с памятью при обычных и параллельных вычислениях была осознана достаточно давно. Циклы «for» представляют собой именно ту часть текста программного обеспечения, которая ответственна за использование массивов, а, значит, определяют объем памяти, требуемой приложению и количество обращений к ней, что в свою очередь напрямую влияет на скорость выполнения программ и энергопотребление вычислительных устройств. Последнее является одной из актуальных проблем при проектировании микросхем, переносных и стационарных вычислительных машин. Операция объединения циклов в исходном тексте описания встроенной системы или программы позволяет уменьшить количество обращений к ОЗУ и его объем [2, 3, 4].

В работах [2, 3] рассматривается алгоритм объединения одномерных и многомерных циклов с целью снижения энергопотребления встроенных систем. Данный алгоритм использует графы для представления зависимостей между циклами в тексте программы и целочисленное линейное программирование для преобразования исходного графа в граф, на основе которого создается текст программы, содержащей объединенный цикл. В работах [2, 3] приводятся примеры объединения циклов с помощью рассмотренного алгоритма, которые показывают, что он не позволяет объединить все возможные циклы. Объединить все циклы возможно с помощью подхода, предлагаемого в данной статье.

Большинство работ по преобразованиям циклов фокусируется на распараллеливании вычислений, в основном, вложенных циклов и не рассматривает данные преобразования для целей снижения энергопотребления вычислительных систем (см., например, классическую работу Alain Darte [5] или работу 2010 г. по распараллеливанию вычислений в циклах в компиляторе GCC [6]).

Предлагаемый в данной статье подход к объединению многомерных циклов разработан для целей снижения энергопотребления встроенных систем, содержащих иерархию памяти (регистры процессора, кэш, ОЗУ), и распараллеливание вычислений в циклах не входит в круг задач работы.

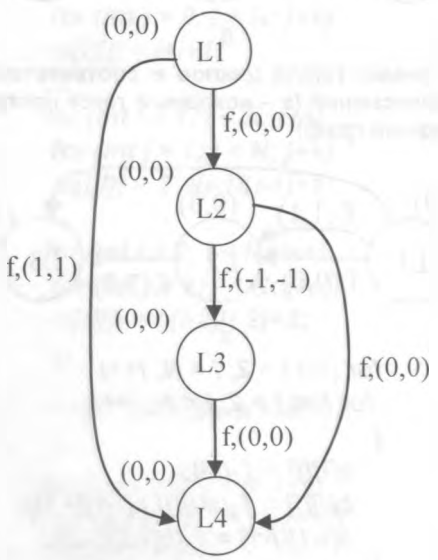
1. Объединение многомерных циклов

Объединение многомерных циклов основано на представлении исходного текста программы в виде графов и их преобразовании. 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.

```
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. 1. Пример текста циклов и соответствующего графа вычислений (а – исходный текст программы, б – граф вычислений)

Его вершины L1 и L2 соответствуют вычислению элементов массивов a1 и a2. 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 $a2[i_1,i_2,\ldots,i_n]$ вычисляются при помощи элементов массива $a1[i_1-d_1,i_2-d_2,\ldots,i_n-d_n]$. Вес, соответствующий i_k итерационной переменной, будет вычисляться, как разница k-х значений индексов для элементов массива a2 и a1: $i_k-(i_k-d_k)=d_k$. 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.

В работе [7] 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.

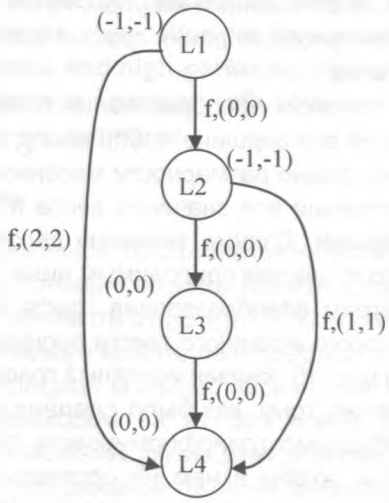
Рассмотрим преобразования графа на примере некоторого исходного текста программы на рис. 1а. На рис. 1б показан исходный граф.

Аналогично тому, как было сделано в работе [7], необходимо трансформировать граф таким образом, чтобы в нём не осталось f-дуг с отрицательными значениями 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. 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. 2 показан конечный преобразованный граф и текст объединённого цикла. Вес вершин графа используется при формировании текста объединённого цикла. Теперь рассмотрим объединение циклов, если между ними присутствует связь по выходу. На рис. 3а представлен некоторый исходный текст программы, содержащий три цикла. На рис. 3б представлен граф, соответствующие исходному тексту программы на рис. 3а.

```
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]);
}
```

а



б

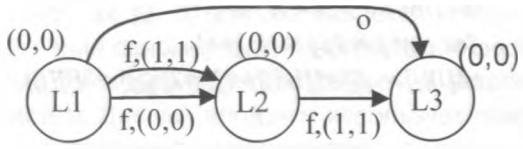
Рис. 2. Пример текста циклов и соответствующего графа вычислений (а – текст объединенного цикла, б – итоговый граф)

Каждой вершине присваивается набор нулевых весов. Веса f-дуг вычисляются так же, как это было сделано выше. Дуга с ярлыком «о» 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. 4а, а текст объединённого цикла – на рис. 4б.

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 приведённых двух равно 1, поэтому первый из набора весов вершины L3 должен увеличиться на 1. При этом веса всех f-дуг, входящих в L3, должны 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.

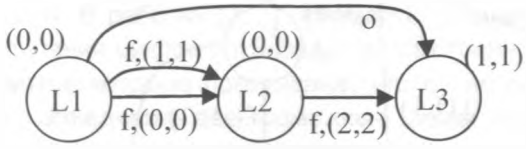
```
for (int i = 0; i < N; i++)
 for (int j = 0; j < N; j++)
  a[i][j] = f1(i,j);
...
for (int i = 1; i < N; i++)
 for (int j = 1; j < N; j++)
  b[i][j] = f2(a[i][j],a[i-1][j-1]);
...
for (int i = 0; i < N - 1; i++)
 for (int j = 0; j < N - 1; j++)
  a[i+1][j+1] = f3(b[i][j]);
```

а



б

Рис. 3. Пример текста циклов и соответствующего графа вычислений (а – исходный текст программы, б – исходный граф)



а

```
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]);
}
```

б

Рис.4. Пример текста циклов и соответствующего графа вычислений (а – итоговый граф, б – текст объединенного цикла)

Количество данных, которые необходимо хранить в кэш памяти или регистрах процессора (будем называть это быстрой памятью), для уменьшения количества обращений к основному запоминающему устройству может быть минимизировано с помощью предлагаемого способа.

Эти циклы можно объединить в один, ничего не меняя в записи вычислений, как это показано на рис. 5г. В этом случае, чтобы уменьшить количество операций чтения из основного запоминающего устройства, необходимо хранить в быстрой памяти значения четырёх 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 только минимум данных, а именно только два значения элементов массивов a[i][j] и b[i+1][j+1], как это показано на рис. 5д. Данный текст цикла получается с помощью преобразований исходного графа циклов на рис. 5б.

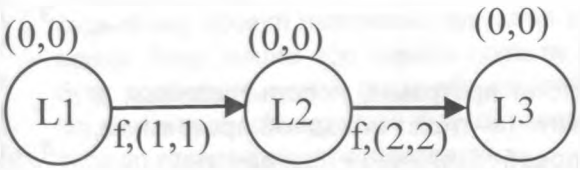
Целью преобразований является минимизация веса всех f-ребер графа, при этом вес каждого ребра должен оставаться неотрицательным. Начиная с вершины L3 и двигаясь к вершине L1, преобразовываем веса всех f-ребер следующим образом. Вес ребра L2→L3 уменьшаем с 2 до 0, при этом вес исходного для данного ребра вершины L2 увеличивается на 2, и вес ребра L1→L2 увеличивается на 2 и становится равным 3. Затем, уменьшаем вес ребра L1→L2 на 3 до 0, увеличивая одновременно вес вершины L1 на 3. Итоговый граф представлен на рис. 5в. Все f-ребра данного графа имеют одинаковый нулевой вес. Это соответствует на рис. 5д тому, что при вычислении значения элемента массива b необходимо хранить в быстрой памяти только одно значение элемента массива a, а при вычислении значения элемента массива c необходимо хранить в быстрой памяти только одно значение элемента массива b.

2. Эксперимент по проверке снижения энергопотребления

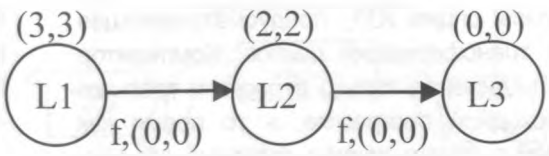
Снижение энергопотребления цифровой системы после объединения циклов в исходном тексте программы было проверено в эксперименте с микроконтроллером MSP430F435 фирмы Texas Instruments. Пусть есть исходный текст приложения, представленный на рис. 6а. С помощью алгоритма объединения циклов исходный текст преобразовывается в текст, приведенный на рис. 6б.

```
...
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;
...
```

а – исходный текст



б – исходный граф



в – преобразованный граф

```
...
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;
}
...
```

г – текст объединённого цикла

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

д – текст объединённого цикла

Рис. 5. Минимизация количества данных для хранения в быстрой памяти

```
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. 6. Тексты программ, использованных при измерении тока потребления микроконтроллера

Для данного микроконтроллера время выполнения программ является одинаковым. В случае выполнения исходной программы ток потребления микроконтроллера составил 680

мкА, для программы с объединённым циклом ток потребления был 538 мкА. В результате операции объединения циклов в тексте программы удалось уменьшить ток потребления микроконтроллера приблизительно на 21%.

```
for (i=0; i<20; i++) a1[i]=i+1;
for (i = 0; i<20; i++) a2[i]=a1[i]+1;
for (i = 0; i<19; i++) a3[i]=a2[i+1];
a3[19]=...;
for (i = 1; i<20; i++) a4[i]=a1[i-1]+a3[i];
a4[0]=...;
for (i = 0; i<20; i++) a5[i]=a2[i]+a3[i]+a4[i];
```

а

```
...
for (i = 2; i<20; i++) {
  a1[i] = i + 1;
  a2[i] = a1[i] + 1;
  a3[i-1] = a2[i];
  a4[i-1] = a1[i-2] + a3[i-1];
  a5[i-1] = a2[i-1] + a3[i-1] + a4[i-1];
}
...
```

б

Рис. 7. Тексты программ, использованных в эксперименте (а – текст исходной программы, б – текст преобразованной программы)

Исполняемые файлы обеих программ были получены с помощью C++ Compiler Professional Edition for Windows фирмы Intel. При компиляции использовалась опция /O3, предусматривающая выполнение трансформаций циклов. Компилятор произвел объединение только второго и третьего циклов в исходной программе, в то время как предлагаемый в статье подход позволил объединить все пять. Время выполнения исходной программы составило 0,047 с, а программы, содержащей объединённый цикл, - 0,015 с.

Выводы

Предлагаемый алгоритм позволяет объединять циклы со связями по данным и выходу одновременно, минимизировать количество данных, которые необходимо хранить в регистрах процессора или кэш-памяти в процессе вычислений. Метод, предложенный в [3], позволяет объединить только часть циклов в тестовом примере, в то время как алгоритм, предлагаемый в данной работе, позволяет добиться лучших результатов - объединить все циклы. В результате объединения циклов, выполненного согласно предлагаемому алгоритму, можно существенно сократить количество операций обращения к ОЗУ и уменьшить его размер. Проведенные эксперименты подтверждают, что после операции объединения циклов уменьшилась величина тока потребления микроконтроллера, а также уменьшилось время выполнения тестовой программы.

Литература

1. International Technology Roadmap for Semiconductors. http://public.itrs.net/
2. *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.
3. *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.
4. *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.
5. *Darte A.* On the complexity of loop fusion // Parallel Computing. – Amsterdam: Elsevier, 2000. – Vol. 26, № 9. – P. 1175-1193.
6. *Trifunovic K.*, Cohen A., Edelsohn D., Feng L., Grosser T., Jagasia H., Ladelsky R., Pop S., Sjodin J., Upadrasta R. GRAPHITE Two Years After. First Lessons Learned From Real-World Polyhedral Compilation // 2010. - P. 16. - URL: http://gcc.gnu.org/wiki/Graphite?action=AttachFile&do=view&target=graphite2y.pdf.
7. *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. 02-12.

Институт проблем моделирования в энергетике НАН Украины, г. Киев