НЕКОТОРЫЕ ТРУДНОСТИ АВТОМАТИЧЕСКОГО РАСПАРАЛЛЕЛИВАНИЯ И ПУТИ ИХ ПРЕОДОЛЕНИЯ

***Климов Аркадий Валентинович**, без уч.ст.*
Институт проблем проектирования в микроэлектронике (ИППМ) РАН

На примере классического алгоритма Флойда-Уоршелла[1] показаны трудности автоматического распараллеливания, связанные с наличием «лишних» формальных зависимостей. Рассматриваются разные способы преодоление этих трудностей: в системе ДВОР [2] – через выяснение в диалоге с пользователем, что эту зависимость следует игнорировать, в потоковой системе БУРАН [5] – зависимость остается, но влияет незначительно, поскольку и остающегося параллелизма вполне достаточно. Однако, и то и другое не панацея, а нужны более гибкие языковые формы для программирования, позволяющие писать исходный код без «лишних» зависимостей.

***Ключевые слова:** автоматическое распараллеливание, анализ зависимостей, гнезда циклов, потоковая модель вычислений, параллельное программирование.*

Одна из основных трудностей в работе программ автоматического распараллеливания связана с наличием ложных зависимостей между итерациями. Они только кажутся реальными зависимостями и потому препятствуют принятию решения о возможности распараллеливания цикла, но на самом деле таковыми не являются. Зависимость между итерациями – это два обращения к одной и той же ячейке в разных итерациях цикла, хотя бы одно из которых является записью. Наличие хотя бы одной такой зависимости не позволяет объявить цикл параллельным, поскольку перестановка двух зависимых итераций может повлиять на результат. Однако это не означает, что на распараллеливании такого цикла следует поставить крест. Рассмотрим в качестве примера классический алгоритм Флойда-Уоршелла[1], часто называемый просто алгоритмом Флойда.

Алгоритм имеет на входе матрицу A длин дуг графа, а на выходе выдает матрицу B длин кратчайших путей. Предполагается, что на месте отсутствующей дуги в матрице A стоит ∞ (бесконечность), или очень большое число. Идея метода состоит в последовательном построении матрицы $W^{(k)}$, k=0,1,…,n, представляющей кратчайшие расстояния при условии, что в качестве промежуточных могут использоваться только первые *k* вершин графа. При этом $W^{(0)}$=A, B=$W^{(n)}$. Имея $W^{(k-1)}$, для вычисления $W^{(k)}_{i,j}$ надо посмотреть, проходит ли искомый кратчайший путь через вершину *k*: если да, то он равен сумме $W^{(k-1)}_{i,k}$+ $W^{(k-1)}_{k,j}$, в противном случае $W^{(k-1)}_{i,j}$. А в общем случае надо взять минимум из этих двух значений. Таким образом, алгоритм может быть записан в форме гнезда циклов:

```
for k = 1 to n
  for i = 1 to n
    for j = 1 to n
      W[i][j] = min(W[i][j], W[i][k] + W[k][j])
```

Внимательный читатель заметит, что эта программа не вполне отвечает вышеприведенному словесному описанию: иногда слагаемые W[i][k] и W[k][j] являются элементами массива W, вычисленными и записанными на итерации k, а не k-1. Именно, при j>k или i>k соответственно. Но ничего страшного: нетрудно доказать, что эти значения на итерации k изменены не были, приняв во внимание, что диагональный элемент W[k][k] неотрицателен (что обеспечивается отсутствием в исходном графе циклов отрицательной длины) [3].

Однако для штатного распараллеливающего компилятора, который доказывать такое не умеет, это будет сигнал, что два внутренних цикла распараллелить нельзя. Рассмотрим три подхода к преодолению данной трудности.

1. Информируем компилятор об отсутствии некоторых зависимостей.

Тем или иным способом программист информирует компилятор о том, что чтения W[i][k] и W[k][j] не порождают межитерационных зависимостей в циклах по i и по j. Например, система ДВОР (Диалоговый высокоуровневый оптимизирующий распараллеливатель программ), разрабатываемая группой Б.Я.Штейнберга [2], обратится к пользователю с вопросом типа: «являются ли элементы (исходной) матрицы W неотрицательными». Получив на него утвердительный ответ, компилятор «поймет», что межитерационных зависимой в циклах по i и j нет, и построит код с распараллеливанием циклов по i и j [3].

2. Транслируем в потоковую модель вычислений, сохраняя зависимости.

Распараллеливающий транслятор [4] строит потоковый граф и генерирует код на потоковом языке DFL, выполняющийся на специальной аппаратуре параллельной потоковой вычислительной системы БУРАН [5]. При этом каждый оператор выполняется лишь после того, как будут выполнены все операторы, вычисляющие входные значения для данного оператора и эти значения будут доставлены к месту выполнения данного оператора. Несмотря на наличие «лишних» зависимостей, в задаче остается достаточно много параллелизма. В частности, вычисление k-й итерации может проходить в следующей последовательности (Рис.1):

- Левый верхний угол (i<k, j<k) параллельно – (I).
- Крест (i=k or j=k), причем сначала центр (i=k, j=k) – (X), потом остальные элементы креста в параллель – (II).
- Все остальные элементы параллельно – (III).

Это лишь возможный порядок. Он может быть и другим, например первая область для k+1-й итерации может считаться параллельно с третьей областью для k-й итерации (после определения значений на ее краях внутреннего угла). Важно отметить, что компилятор параллельный порядок вычислений не планирует: он определяется динамически в соответствии с принципом готовности данных.

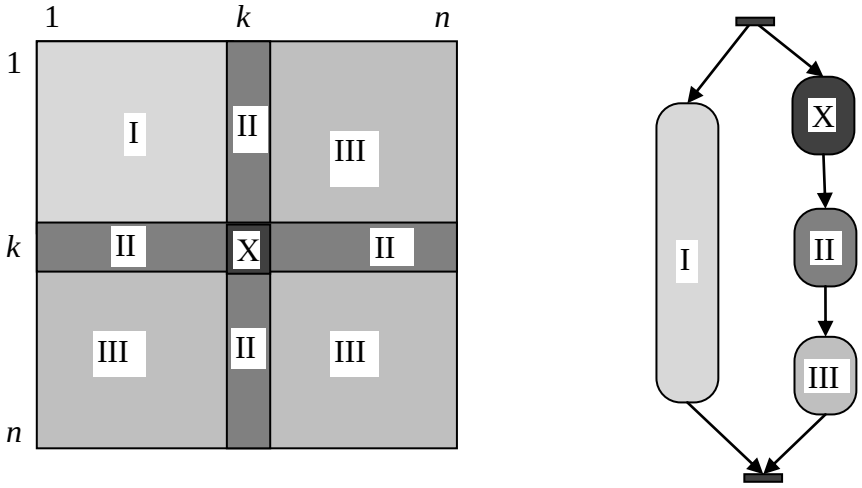

Рис.1. Разбиение матрицы *W* на области, выполняемые в соответствии с определенным частичным порядком (справа).

Нам удалось на модели получить ускорение в 36 раз на 64-х процессорных ядрах (эффективность 0.56) для графа из 16 вершин (по сравнению с выполнением той же задачи на 1-м ядре). Характерно, что процесс шел одновременно на нескольких (3-х) соседних итерациях по индексу k. Элементы матрицы были распределены равномерно циклически: на каждом ядре по 4 с шагом 8 по направлениям i и j. При блочном распределении (квадратами 2x2) результат оказался хуже: ускорение только 26 раз (эффективноть 0.41). Основным источником снижения эффективности является неравномерность вычислительной нагрузки: на области, пересекающие «крест» (i=k or j=k), падает больше работы, чем на удаленные от «креста». При циклическом распределении «тяжелые» элементы более равномерно раскидываются по ядрам, хотя при этом и увеличивается нагрузка на сеть.

3. Программируем изначально параллельно.

Думается, что ни один из путей 1 или 2 не может считаться панацеей: то, что удалось в этой задаче, может и не сработать в других. Данный пример является прекрасной иллюстрацией известного тезиса о том, что последовательные алгоритмы часто испорчены искусственными деталями, связанными именно с их последовательностью. В данном случае автор исходной циклической программы сознательно или бессознательно отклонился от первичной идеи, выраженной в рекуррентном правиле вычисления матрицы $W^{(k)}$, в пользу более «красивого» и «экономного» последовательного кода. Правильнее было бы выразить ее кодом, использующим дополнительный массив (двумерный или два одномерных), чтобы при вычислении $W^{(k)}$ использовались только значения из $W^{(k-1)}$. Тогда автоматическое распараллеливание было бы и простым и более эффективным. И это было бы фактически параллельным программированием в последовательном языке. Наконец, можно ожидать появление в будущих версиях фортрана специальных средств управления (с семантикой типа «все записи в такой-то массив в таком-то блоке должны быть исполнены в конце блока»), которые позволят явно указать, что правые части всегда апеллируют к значениям, записанным на предыдущей итерации.

1. Кристофидес Н. Теория графов. Алгоритмический подход. - М.: Мир, 1978, 429 с.
2. Штейнберг Б.Я., Абрамов А.А., Алымова Е.В., Баглий А.П., Гуда С.А., Дубров Д.В., Кравченко Е.Н., Морылев Р.И., Нис З.Я., Петренко В.В., Полуян С.В., Скиба И.С., Шаповалов В.Н., Штейнберг О.Б., Штейнберг Р.Б., Юрушкин М. Диалоговый высокоуровневый автоматический распараллеливатель (ДВОР). Научный сервис в сети Интернет: Труды Всероссийской суперкомпьютерной конференции (20-26 сентября 2010, г. Новороссийск). М.: Изд-во МГУ, 2010, с. 71-75
3. Штейнберг Б. Я., Гуда С. А., Морылев Р. И., Баглий А. П., Скиба И. С. Анализ информационных зависимостей в ДВОР. РАСО'2012/ Труды международной конференции «Параллельные вычисления и задачи управления». М., 24-26 октября 2012 г., ИПУ РАН. (в печати?)
4. Климов Арк.В. Трансляция последовательной программы в потоковый язык как способ распараллеливания. Материалы Международной научно-технической конференции «Суперкомпьютерные технологии: разработка, программирование, применение», Дивноморское, 27 сент.-2 окт. 2010, с. 246-250.
5. Стемпковский А.Л., Левченко Н.Н., Окунев С.А., Цветков В.В. Параллельная потоковая вычислительная система — дальнейшее развитие архитектуры и структурной организации вычислительной системы с автоматическим распределением ресурсов. // Информационные технологии, № 10, 2008, с. 2-7.