АВТОМАТИЗАЦИЯ РАСПАРАЛЛЕЛИВАНИЯ АЛГОРИТМИЧЕСКИХ МОДЕЛЕЙ

Р. А. Крылов (Санкт-Петербург)

Введение

Алгоритмические модели – алгоритмы, структура которых подобна структуре причинно-следственных связей в сценарии моделируемого объекта, поэтому они должны отражать естественный параллелизм процесса. При отображении причинно-следственных связей информационными связями [1] в алгоритме структура информационных связей будет соответствовать операционным связям, то есть такой алгоритм будет максимально распараллелен. Однако это требует специальных методов представления параллельных алгоритмов, которыми не все владеют. Многие модели уже описаны в виде последовательных алгоритмов, поэтому встает вопрос восстановления их параллельного вида для решения на распределенных вычислительных ресурсах и распознавания исходного сценария процесса.

Распараллеливание и распределение вычислительных мощностей является моментом, затрудняющим написание программы. Поэтому распределение вычислительных мощностей должно выполняться самой операционной системой, а распараллеливание программы желательно было бы переложить на специальные утилиты. Прикладного программиста интересует в первую очередь решение задачи, а во вторую – ее эффективность. Поэтому зачастую в многопроцессорные ЭВМ и суперкомпьютерные кластеры запускаются программы без распараллеливания, что приводит к неэффективному использованию вычислительных мощностей. Выше изложенное определяет необходимость разработки средств автоматизации распараллеливания программ, написанных в последовательной форме. Это позволит не отказываться от существующих и хорошо себя зарекомендовавших технологий разработки последовательных программ, повысит эффективность использования многопроцессорных вычислительных систем.

Целью данной работы является разработка средств автоматизации распараллеливания последовательных программ, без учета ограничений аппаратной среды. В данной работе в качестве таких программ исследуются программы на языке С. Разработанные принципы могут быть применены для создания средств распараллеливания программ и на других языках.

Идея подхода

Идея рассматриваемого подхода проста. Если знать все информационные связи операторов программы и уметь выявлять множества не зависящих информационно друг от друга операторов, то их можно запускать параллельно. Данная идея восходит к ярусно-параллельным графам для планирования решения сложных задач в сети ЭВМ [2]. Эта же идея использовалась для планирования вычислений на алгоритмических сетях [1], которая позволила рассмотреть использование алгоритмических сетей для организации параллельных вычислений [3]. При распараллеливании программы не учитываются возможные аппаратные ограничения.

Построение информационных зависимостей проще выполнять для линейной части программы, следовательно, необходимо научиться свертывать разветвленные участки программы, циклы, сохраняя информацию об информационных связях между получаемыми блоками. Возможен также и другой подход: выделять каждый участок с ветвлением или вложенным циклом в подпрограммы. Подобный подход предлагался в [3]. Результат получается тот же – линейная программа, но там это, в основном, определялось особенностями используемого языка представления алгоритмов и требовани-

ем, чтобы каждая переменная алгоритма имела однозначную интерпретацию в терминах предметной области.

Стиль программирования может оказать существенное влияние на сложность процедуры свертки участков программы. Наиболее просто это делать для программ, использующих только конструкции структурного программирования: следование, ветвление, цикл. Как показано в [4], множество структурных алгоритмов равномощно множеству всех алгоритмов, и имеется процедура, позволяющая преобразовать любой произвольный алгоритм в эквивалентный ему по реализуемой функции структурный. Главной особенностью данных конструкций, облегчающих свертку, является наличие только одной входной и только одной выходной дуги.

Будем считать, что оператору должны предшествовать все те операторы, результаты которых являются для него входными, и все операторы, использующие его результаты, должны следовать за ним. Далее ранее свернутые операторы могут быть раскрыты, и для их тела процедура может быть повторена и т.д. Отслеживать преобразование программ в параллельный вид легче всего на каком-либо графическом представлении алгоритмов. В качестве графического представления используются параллельные граф-схемы алгоритмов (ПГСА), и процесс распараллеливания последовательной программы визуально представляется как превращение обычной граф-схемы в параллельную. При описываемом подходе структура управляющих связей в ПГСА будет изоморфна структуре информационных связей между операторами. Однако такая структура будет избыточна. Так как, если некоторый оператор выполняется после какого-либо другого, и между ними выполняется путь, включающий ряд других операторов, то информационную связь между ними можно опустить, поскольку к моменту его выполнения все исходные данные будут уже готовы. Иначе можно редуцировать все транзитивные замыкания путей в полученной ПГСА. Это позволяет сделать представление более наглядным и упростить формирование параллельных процессов.

Формализация

Рассматриваемый подход имеет общий характер, присущий большинству процедурно-ориентированных языков программирования. В каждом из таких языков можно выделить составные операторы, которые могут встречаться где угодно в программе, в том числе, внутри операторов цикла и внутри выбирающих операторов.

Составной оператор представляет собой набор последовательно выполняемых операторов. В случае непараллельного кода каждый оператор выполняется сразу после выполнения предыдущего: $op_1 \to op_2 \to \ldots \to op_n$, то есть выполняется обязательное предшествование i-го оператора j-му, при $i<j$. В общем случае отношение полного порядка между операторами не требуется, и оно может быть ослаблено в пользу параллельно выполняемых операторов.

Руководствуясь синтаксисом и семантикой языка, для каждого оператора находятся входные и выходные переменные. Входными переменными назовем все используемые в операторе переменные, то есть все переменные, которые должны быть объявлены до выполнения оператора. Выходными назовем лишь те, которые меняются в процессе выполнения оператора. Таким образом, все выходные переменные являются также и входными, то есть $out(op) \subseteq in(op)$.

Используя множества входных и выходных переменных, формируется бинарное отношение между операторами:

$$P_{ij} := i < j \wedge (out(op_i) \cap in(op_j) \neq \varnothing) \vee (in(op_i) \cap out(op_j) \neq \varnothing).$$

Элементы полученной треугольной матрицы *P* могут рассматриваться как дуги ориентированного графа, в вершинах которого стоят выполняемые операторы. Построив транзитивное замыкание этого графа, получаем матрицу полных связей – отношение частичного порядка на множестве операторов:

$$B_{ij} := \exists k_1,...k_m : P_{ik_1} \wedge P_{k_1k_2} \wedge ... \wedge P_{k_{m-1}k_m} \wedge P_{k_mj},$$

т. е., существует путь из *i*-й вершины в *j*-ю. B_{ij} означает, что *i*-й оператор должен быть выполнен до *j*-го. Так же строится минимальная матрица связей:

$$C_{ij} := B_{ij} \wedge \left\{ \forall k \in 1..n \ \overline{(B_{ik} \wedge B_{kj})} \right\},$$

т. е., не существует другого пути из *i*-й вершины в *j*-ю, кроме как напрямую. В матрице *C* отображены лишь основные связи. Смысл ее в том, что оператору не требуется дожидаться всех предшественников, а лишь некоторых. Дожидаясь окончания выполнения какого-либо из предшественников, мы можем гарантировать, что его предшественники, в свою очередь, уже будут завершены, дожидаться их не имеет смысла. Таким образом, упрощая связи предшествования, алгоритм программы не будет нарушен при выполнении.

После того, как известны все зависимости выполнения между операторами, требуется априори распределить выполнение по потокам. Дело в том, что в рамках данной работы мы не располагаем информацией о времени выполнения каждого из операторов, а лишь руководствуемся порядком выполнений. Мы могли бы в каждой точке распараллеливания вычислительных потоков создавать новые потоки, но мы задаемся целью уменьшить число потоков, оставив при этом максимально распараллеленную схему.

Операторы расставляются по потокам за один проход. Изначально подразумеваем, что все потоки свободны, и число их ограничено лишь количеством операторов. Перебирая в порядке очередности все операторы, расставляем их следующим образом:

1) если у очередного оператора нет предшественников, он ставится в новый поток;

2) если предшественники есть, пытаемся поместить в один из потоков, где находится какой-либо из предшественников (при условии, что после этого предшественника еще не поставлен оператор);

3) в противном случае пытаемся поставить в каждый поток, начиная с первого, (при условии, что в этом потоке нет оператора, ждущего своего выполнения).

Заключение

Для вышеописанного подхода была создана программная система, основным предназначением которой являются: исследование программного кода на языке С, автоматизация возможностей распараллеливания кода и исследование параллельных схем, базирующихся на стандартных конструкциях. Данная программная система является системой советчиком, позволяющим пользователю отредактировать текст программы для его параллельной реализации. Возможно и другое использование рассматриваемого подхода, а именно – декомпозиция программ для их распределенного расчета. В данном случае это подробно не рассматривается, но доказано, что декомпозиция программы по ее параллельным ветвям обеспечивает минимум необходимых информационных взаимодействий между частями программы. Задачей дальнейших исследований является не только распространение предложенного подхода на другие языки моделирования и создание системы оптимизации структуры параллельной программы, учитывающей особенности структуры используемых аппаратных средств, но и их применение в сетевых распределенных вычислениях.

Литература

1. **Иванищев В.В., Марлей В.Е.** Основы теории алгоритмических сетей. – СПб.: СПбГТУ, 2000. – 180 с.
2. **Поспелов Д.А.** Введение в теорию вычислительных систем. – М: Советское радио, 1972. – 280 с.
3. **Марлей В.Е.** Алгоритмические сети и параллельные вычисления//Труды СПИИ-РАН. – СПб., 2002. – Т. 2. – Вып. 1. – С. 114–124.
4. **Дал У., Дейкстра Э., Хоар К.** Структурное программирование. – М.: Мир, 1976. – 90 с.