МОДЕЛЬ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ ВИЗУАЛЬНОГО ГРАФ-ОРИЕНТИРОВАННОГО ЯЗЫКА

Востокин С.В.

В настоящее время в области параллельного программирования произошли кардинальные изменения. Высокопроизводительные SMP-компьютеры, 100MB сети Ethernet стали практически стандартным офисным оборудованием. Однако средствами разработки и программным обеспечением по-прежнему могут пользоваться только программисты-системщики высокой квалификации. Для преодоления сложностей, возникающих при разработке параллельных программ, предлагается использовать визуальный граф-ориентированный язык программирования Graph+.

Разработка модели языка Graph+ проводилась исходя из следующих основных технических требований.

1. Простота модели, минимум необходимых средств синхронизации и управления.
2. Использование графической модели как средства абстрагирования от конкретной реализации программы.
3. Использование параллелизма в явном виде.
4. Реализация языка как надстройки над произвольным базовым алгоритмическим языком.
5. Возможность исполнения программ в архитектурах с разделяемой и с распределенной памятью.
6. Модель должна обеспечивать эффективность генерируемого кода.
7. Система программирования должна представлять из себя генератор текстов на алгоритмическом языке с использованием системных средств синхронизации из высокоуровневой спецификации на разрабатываемом языке.

В предлагаемой модели программирования исходим из того, что алгоритм можно составить из нескольких функций и процедур, взаимодействующих между собой через общие переменные. Передача значений переменных при вызове происходит по ссылке. То есть в функцию или процедуру передается указатель на используемые данные. В качестве выражения условного оператора в нашей модели используется логическая функция, в которую также передаются указатели на переменные.

Данное представление программ, во-первых, позволяет абстрагироваться от языка реализации процедуры или функции, а во-вторых, что более существенно для модели, позволяет изобразить программу в виде ориентированного графа. Рассмотрим граф (рис.1) и его интерпретацию.

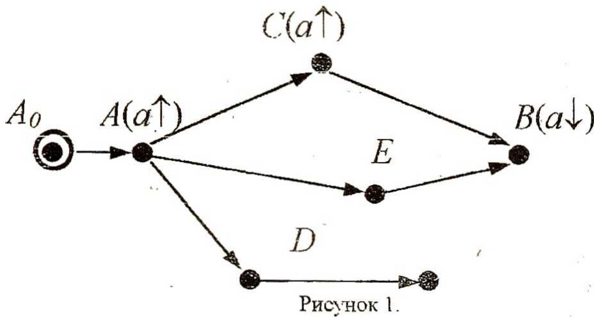

Рисунок 1.

Граф является ориентированным и инициальным. Его начальная вершина (A_0) является точкой входа в программе. Вершины графа обозначают процедуры, а сам граф — передачу управления от одной процедуры к другой. Вычисления останавливаются, когда выполнится вершина, не имеющая исходящих дуг. Очевидно, что вычислительным алгоритмам будут соответствовать лишь те графы, у которых: а) любая вершина достижима из начальной; б) для каждой не концевой вершины существует некоторая достижимая концевая вершина графа. Будем также считать, что выбор дуги, по которой происходит передача управления, реализуется согласно внутренней логики работы вершины.

Для описания взаимодействия по данным необходима дополнительная информация о том, как в процедурах используются значения, переданные по ссылке. В нашей модели (в отличие от традиционных процедурных языков) такая информация задается в явном виде как признак использования переменных. На графе (рис.1) мы использовали следующие обозначения: $a\uparrow$—переменной присваивается новое значение; $a\downarrow$ — значение переменной используется в процедуре, но не изменяется; $a\downarrow\uparrow$ — комбинация двух предыдущих (начальное значение используется для вычисления нового значения).

Пусть теперь требуется организовать вычисления разных процедур на разных вычислительных устройствах, каждое из которых имеет собственную локальную память и отдельный процессор. Взаимодействие между устройствами организуется посредством пересылки данных. Такая ситуация характерна для кластерных архитектур. Несложно убедиться, что для графа (рис.1) взаимодействие через общие переменные можно смоделировать в распределенной архитектуре исполнения. Рассмотрим некоторую переменную a из набора переменных, образующих общую память программы. Значения, хранящиеся в переменной a, могут передаваться между любыми парами (A,B) вершин графа по маршрутам, для которых выполняются следующие условия:

1) $A(a\uparrow)$ или $A(a\downarrow\uparrow)$, $B(a\downarrow)$ или $B(a\downarrow\uparrow)$;

2) B достижима из A;

3) нет вершины $C(a\uparrow)$ или $C(a\downarrow\uparrow)$, принадлежащей маршруту из A в B.

Невыполнение последнего правила приведет к замещению значения, вычисленного в A, новым значением, вычисленным в C. Для графа (рис.1) такой маршрут проходит через вершину E. Во всех вершинах, по которым проходят указанные маршруты, заводятся локальные копии переменной a. Передача управления реализуется как посылка специального сообщения, в котором: передаются собственно значения переменных и передается вектор признаков, в котором для каждой переменной указывается передается ли ее значение в данном сообщении.

В вершинах организуется логика управления признаками. Признак передачи устанавливается для переменной a в вершине A, если $A(a\uparrow)$ или $A(a\downarrow\uparrow)$. Признак сбрасывается, если собираемся выполнить переход по дуге в вершину D, в которой переменная a не используется и не буферизуется. Таким образом, приходим к представлению вычислительного процесса в виде объекта, обладающего памятью, который перемещается от одного вычислительного устройства к другому. По аналогии с известной моделью многопоточного мультипрограммирования, данный объект называется вычислительным потоком.

Следующим шагом в построении модели параллельных вычислений является организация одновременного исполнения нескольких вычислительных потоков. В рассматриваемой модели параллелизм реализуется посредством механизма многопоточности. Граф параллельной программы (рис.2) составляется из нескольких подграфов подобных графу на рис.1, каждый из которых имеет инициальную вершину. Одна из инициальных вершин (вершина A_0) является точкой входа. С нее начинается исполнение главного потока программы. Между подграфами имеется связь (дуга AD). Дуга AD предназначена для порождения новых потоков. Когда некоторый поток достигает вершины A, из которой исходит эта дуга, в подграфе DGF порождается новый поток.

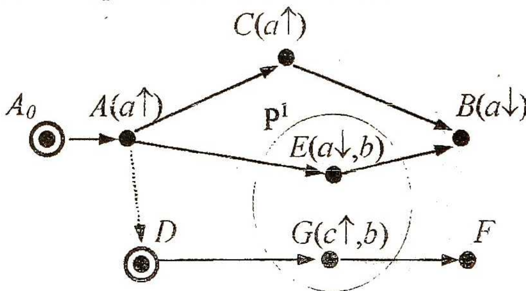

Рисунок 2.

Потоки завершаются, когда достигают концевых вершин в своих подграфах. Все потоки исполняются независимо друг от друга и имеют одинаковый приоритет. Однако никакой поток не может бесконечно долго оставаться в некоторой вершине. Некоторый поток может запустить на исполнение другой поток. В данной модели не предусмотрено средств для приостановки исполнения потоков.

Последнее, что необходимо для описания реальных параллельных алгоритмов это механизм передачи информации из одного потока в другой. Этот механизм реализуется при помощи виртуальных процессоров. В терминах графа (рис.2) виртуальный процессор P^1— это множество вершин, которые могут принадлежать разным подграфам потоков и множество разделяемых потоками переменных (b). Виртуальный процессор в каждый момент времени может исполнять только один поток. Таким образом, он работает как критическая секция. Если внутри виртуального процессора оказывается дуга, порождающая новый поток, то он не активируется до тех пор, пока исходный поток не покинет процессор или не достигнет концевой вершины.

Если некоторый поток хочет передать информацию в другой, он делает запись в разделяемые переменные. Любой другой поток, который войдет в виртуальный процессор, может считать записанное ранее значение. На рис.2 потоки обмениваются информацией через разделяемую переменную b. Для этого поток A_0 переписывает значение, полученное в своей локальной переменной a в разделяемую переменную b. Значение переменной b, затем копируется в локальную переменную c потоком D.

Так как в процессе исполнения графа (рис.2) одновременно активны несколько потоков, использующих общие ресурсы, то при реализации отображения возможно возникновение коммуникационных тупиков и отталкивания. Тупик может возникнуть в следующей ситуации (рис.3).

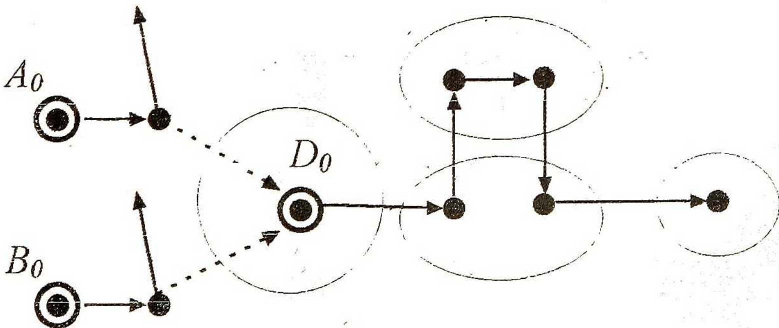

Рисунок 3.

Подграф потока D_0 может принадлежать нескольким виртуальным процессорам. Следовательно, в подграфе D_0 может одновременно исполняться несколько потоков. Если в подграфе есть цикл, проходящий по нескольким виртуальным процессорам, то возможна ситуация, когда каждый поток будет ждать освобождения буферизуемых переменных и ни один не сможет продолжить исполнение. Отталкивание возможно, например, при отображении графа (рис.2), если по какой-либо причине виртуальный процессор P^1 будет обрабатывать только сообщения от потока A_0.

При разрешении таких ситуаций исходим из простоты трансляции модели и эффективности исполнения. Для исключения отталкивания используется циклический алгоритм опроса буферов. Для исключения коммуникационных тупиков используется дополнительная буферизация. Кроме того, новь порожденный поток, который в момент образования не имеет внутренних переменных, задерживается до момента освобождения входного буфера виртуального процессора. Если в то время, пока поток находится в задержанном состоянии в начальной вершине, поступает запрос на порождение нового потока, этот запрос игнорируется.

На основе представленной модели была разработана среда программирования (IDE) языка Graph+. Она включает графический редактор, компилятор диаграмм в платформо-независимый код, генератор платформо-зависимого кода. IDE Graph+ представляет собой надстройку над произвольной средой разработки и произвольным алгоритмическим языком программирования. В настоящей опытной реализации языка Graph+ используется язык C/C++, IDE MS Visual Studio и IDE C++ Builder.

Экспериментальное сравнение тестовой программы, построенной по диаграммам языка Graph+, с эталонной программой, построенной вручную, показало, что при решении практических задач большой размерности, для которых действительно актуально распараллеливание, разница в скорости счета между тестовой и эталонной программой незначительна. Тем самым подтверждена возможность практического использования предложенной графической модели программирования параллельных процессов.