ИСПОЛЬЗОВАНИЕ АСИНХРОННО ПОСТУПАЮЩИХ ДАННЫХ В ПОТОКОВОЙ МОДЕЛИ ВЫЧИСЛЕНИЙ*

А.И. Легалов

КГТУ, Красноярск

Введение

Асинхронное управление широко используется в параллельном программировании. На его основе разработаны различные модели вычислений. Одной из основополагающих работ является книга Хоара [1], в которой описаны базовые методы асинхронного взаимодействия и подведен соответствующий теоретический фундамент. В схемах потока данных (СПД) тоже используется асинхронное управление, которое связывается с моментами готовности операндов той или иной выполняемой операции [2]. Это позволяет говорить о том, что данный подход к вычислениям в потенциале обеспечивает достижение максимального параллелизма, зависящего только от информационных связей решаемой задачи.

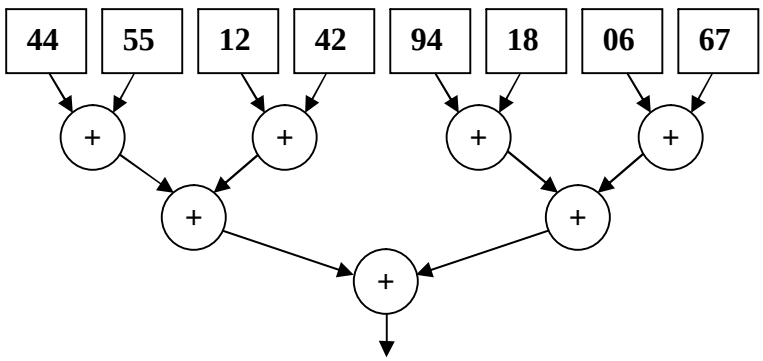


Рис. 1. Суммирование с использованием древовидной свертки

Вместе с тем следует отметить, что известные на данный момент СПД не позволяют эффективно описывать более гибкие методы обработки асинхронных потоков данных, что можно показать на примере нахождения суммы элементов вектора. Большинство потоковых языков используют в вычислениях заранее подготовленный исходный массив. При наличии в этом массиве всех элементов параллельные вычисления можно организовать на основе древовидной свертки, когда в начале суммируются пары из исходного массива, затем результаты, полученные после суммирования этих пар и т.д. до получения окончательного результата. Схема такого

* Работа выполнена при поддержке РФФИ, грант № 05-07-90201.

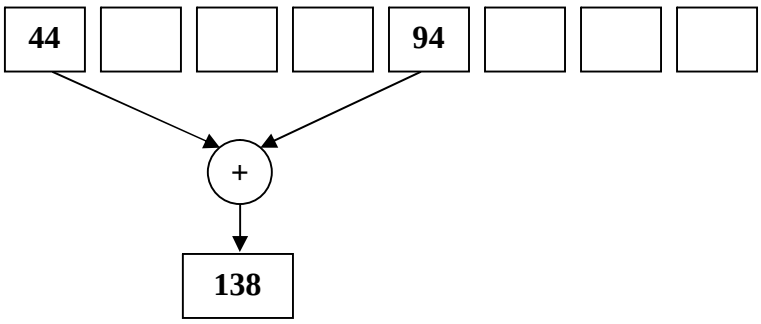


Рис. 2. Возможность суммирования при наличии двух любых элементов

суммирования для массива, состоящего из восьми элементов, представлена на рис. 1.

Следует отметить, что существуют задачи, в которых обрабатываемый массив заранее не известен, а формируется уже в ходе выполнения вычислений. Например, к моменту вызова функции суммирования возможно наличие только 1-го и 5-го элементов массива (рис. 2).

Специфика задачи позволяет не дожидаться всех данных, осуществляя суммирование любых появившихся элементов. Если к окончанию операции суммирования, представленной на рис. 2, в массив поступил только один новый элемент, то его можно сложить с только что вычисленным результатом (рис. 3) до появления других чисел и т.д.

Таким образом, можно оперативно проводить обработку любых поступивших элементов и сформированных промежуточных результатов. Однако подобный метод управления вычислениями в существующих потоковых моделях и языках не поддерживается. Ниже предлагаются механизмы организации данных и методы управления вычислениями, расширяющие возможности асинхронного управления в СПД. Показывается, каким образом предлагаемые понятия расширяют функционально-потоковую модель параллельных вычислений и могут быть реализованы в языке программирования «Пифагор» [3−4].

Специфика асинхронного управления данными

Опираясь на работу Хоара [1], будем считать, что события, возникающие при выполнении параллельных процессов, порождаются последовательно. В СПД эти события связаны с моментами получения значений данных. Интервал времени между порождаемыми событиями в асинхронных моделях не считается существенным фактором. Более важную роль в данном случае играют соотношения между моментами порождения данных и длительностью вычислительных операций. Эти времена могут быть сравнимыми или соотноситься как очень большие и очень малые величины. Опираясь на представленные рассуждения, можно предложить следующую схему вычисления суммы элементов одномерного массива на основе готовности данных. Элементы массива, асинхронно формируемые в

ходе вычислений, образуют очередь, в которую заносятся в порядке появления. При наличии в этой очереди двух или более элементов проводится суммирование образуемых пар с последующей засылкой результата в эту же очередь. Единственный элемент, оставшийся в ходе суммирования, является окончательным результатом. Схема подобной организации вычислений представлена на рис. 4.

Для реализации такого управления в функциональную модель потоковых вычислений предлагается добавить асинхронный список. Его особенностью является упорядочение данных в соответствии с последовательностью их порождения. По сравнению с обычным списком данных, выдающим сигнал готовности после поступления всех его элементов, асинхронный список сигнализирует только о появлении первого элемента.

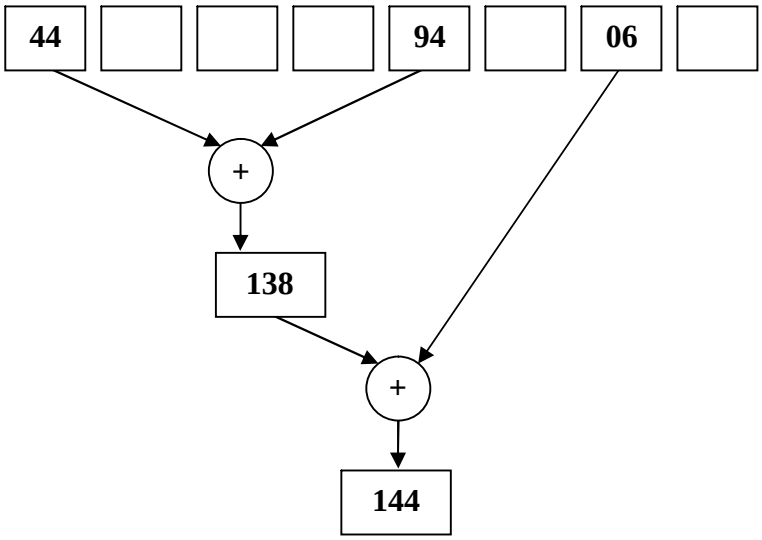


Рис. 3. Суммирование появившегося элемента с вычисленным промежуточным значением

Это накладывает определенную специфику на его интерпретацию. Можно считать, что об асинхронном списке достоверно известно следующее: или он не готов, или содержит как минимум один готовый элемент (а об остальных данных ничего не известно), или является пустым.

Неготовность асинхронного списка определяется отсутствием сигналов, информирующих о наличии сформированных внутри него данных. В этом случае, в соответствии с принципами потокового управления по готовности данных, такой список не может быть обработан. Выдача сигнала подтверждает готовность асинхронного списка к вычислениям, информируя при этом об одной из двух ситуаций: список содержит хотя бы один элемент или он пуст. Тогда определение внутреннего состояния возможно с использованием функции, определяющей длину асинхронного списка.

Длина пустого списка равна нулю. Длина непустого асинхронного списка всегда равна единице. Это объясняется спецификой его интерпре-

тации. Как только внутри списка появляется хотя бы один вычисленный элемент, происходит выдача сигнала, подтверждающего готовность списка к последующей обработке. Даже при одновременном порождении внутри асинхронного списка нескольких элементов будет выдаваться только один сигнал. В этой ситуации первый элемент списка определяется внутренним арбитром, стратегия работы которого не имеет существенного значения.

Наряду с определением длины, над асинхронным списком допустимы операции чтения первого элемента и формирования нового асинхронного списка, не содержащего первый элемент. Это, в соответствии с концепциями событийного управления, позволяет описывать асинхронный параллелизм как процесс обработки последовательно наступающих событий.

Чтобы отличить асинхронный список от обычного списка данных, ис-

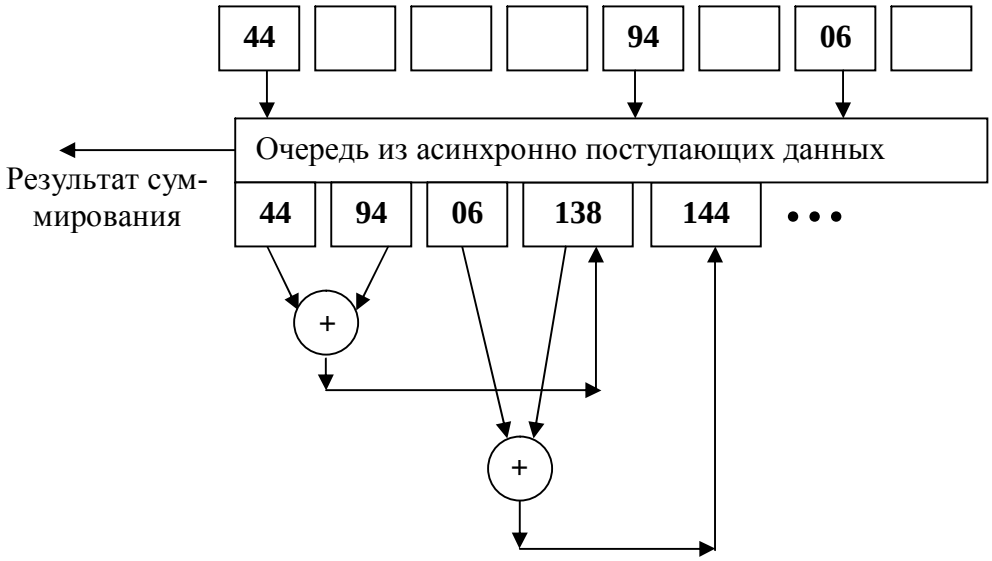


Рис. 4. Суммирование асинхронно поступающих данных

пользуемого в языке «Пифагор», введем для него следующее обозначение: asynch (d_1, d_2, … d_N), где d_1, d_2, … d_N − список элементов, порождаемых в ходе вычислений, проистекающих внутри него, упорядоченный по времени формирования значений. Для обработки этих значений допускается использовать операции выборки первого элемента и создания списка без первого элемента, эквивалентные по синтаксису и семантике, аналогичным операциям списка данных. Таким образом, выделения из списка первого элемента задается следующим образом:

asynch(d_1, d_2, … d_N):1 $\Rightarrow$ d_1 .

Для выделения прочих элементов в отдельный асинхронный список необходимо воспользоваться следующей операцией:

asynch(d_1, d_2, … d_N):-1 $\Rightarrow$ asynch(d_2, … d_N) .

Одновременное использование этих двух операций позволяет выделить и обработать все элементы асинхронного списка. Выполнение этих

операций может продолжиться до получения вместо асинхронного списка пустого списка данных:

asynch(d_1, d_2, d_3, … d_N):[1, -1] $\Rightarrow$ d_1, asynch(d_2, d_3, … d_N,) :[1, -1] $\Rightarrow$ $\Rightarrow$ d_1, d_2, asynch(d_3, … d_N,) :[1, -1] $\Rightarrow$ d_1, d_2, d_3, asynch(d_4, … d_N) :[1, -1] $\Rightarrow$ … … $\Rightarrow$ d_1, d_2, d_3, … d_{N-1}, asynch(d_N):[1, -1] $\Rightarrow$ d_1, d_2, d_3, … d_N, (.).

Прямой доступ к произвольному элементу асинхронного списка невозможен. Если же данные были сформированы заранее, то порядок их следования в асинхронном списке определяется размещением в исходном списке данных.

Дополнительные аксиомы и операции интерпретации функционально-потоковой модели вычислений

Для реализации асинхронного управления в функционально-потоковой модели вычислений [3, 4] введем ряд дополнительных понятий. Пусть имеется асинхронный список asynch (g_1, g_2, … g_n), где g_1, g_2, … g_n – генераторы данных, порождающие величины в произвольной последовательности. Как и список данных, асинхронный список фильтрует (отбрасывает, игнорирует) пустые элементы, не выставляя их очередь данных:

asynch(g_1, .) ≡ asynch(g_1) .

Асинхронные списки можно преобразовывать в другие списки. Преобразование в параллельный список осуществляется по следующему правилу:

asynch(d_1, d_2, … d_N):[] $\Rightarrow$ [d_1, d_2, … d_N] .

В результате на выходе формируется параллельный список, состоящий из множества независимых данных. Порядок следования данных в параллельном списке совпадает с последовательностью их порождения в асинхронном.

Операция преобразования асинхронного списка в список данных определяется аналогичным образом:

asynch(d_1, d_2, …, d_N):() $\Rightarrow$ (d_1, d_2, … d_N) .

Допустимы также обратные преобразования существующих списков в асинхронные, которые осуществляются с использованием существующих аксиом и правил преобразования модели вычислений. Непосредственным является только преобразование списка данных в асинхронный список:

(d_1, d_2, … d_N):asynch $\Rightarrow$ asynch(d_1, d_2, … d_N) .

При преобразовании параллельного и задержанного списков применяются аксиомы модели и языка, определяющие групповые операции [3, 4].

Использование асинхронных списков в языке программирования «Пифагор»

Приведенные выше расширения функционально потоковой модели вычислений можно реализовать в языке программирования «Пифагор». Это позволит писать параллельные программы в несколько ином стиле, при котором учитывается асинхронное поступление данных. В частности, сложение элементов одномерного массива, соответствующее схеме, представленной на рис. 4, может быть реализовано следующим образом:

```
// Функция, возвращающая сумму элементов асинхронного списка
A_VecSum<< funcdef A { // Формат A: asynch(x1, x2, … , xn)
  x1<< A:1;            // Поступивший элемент данных
  tail_1<< A:-1;       // Хвост асинхронного списка
  // Проверка на пустоту остатка списка
  [((tail_1:|, 0):[=, !=]):?]^
  (
     x1, // В списке, только один элемент,  определяющий сумму
     { // Выделение второго аргумента с последующим суммированием
        block {
           x2<< tail_1:1;       // второй аргумент
           s<< (x1,x2):+;       // сумма двух поступивших элементов
           tail_2<< tail_1:-1;  // "хвост" от "хвоста"
           // Рекурсивная обработка оставшихся элементов
           [((tail_2:|, 0):[=, !=]):?]^
           (
              s,
              { asynch( tail_2:[], s):A_VecSum }
           ):. >>break
        }
     }
  ):. >>return;
}
```

Следует отметить, что при поступлении на вход данной функции обычного списка данных вместо асинхронного списка он будет корректно обработан. Это обуславливается эквивалентной интерпретацией операций взятия первого элемента и выделения подсписка без первого элемента для обоих списков.

Интерпретация временных соотношений асинхронной потоковой программы

Использование асинхронных списков позволяет разрабатывать программу, которая, в зависимости от временных соотношений между ее операциями, может интерпретироваться как набор альтернативных алгоритмов, описывающих одну и ту же задачу. Например, если интервал времени Δt_{dat} между порождением данных внутри асинхронного списка больше времени выполнения операции сложения Δt_{add} (от момента чтения данных

из списка до засылки их во вновь формируемый асинхронный список), то суммирование будет выполняться последовательно. Данная ситуация проиллюстрирована на рис. 2 и 3.

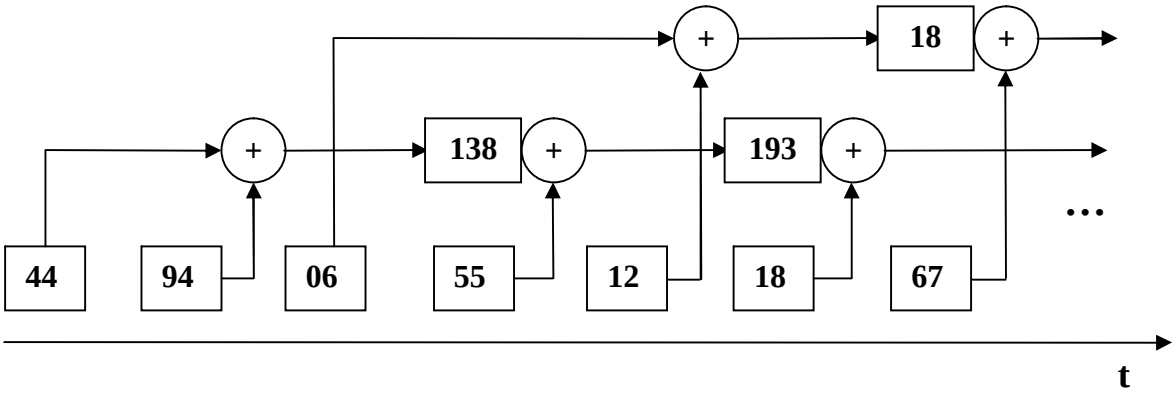


Рис. 5. Формирование двух параллельных потоков при $\Delta t_{add} = 2\Delta t_{dat}$

В том случае, если интервал времени поступления данных станет меньше времени их сложения, начнет появляться параллелизм в выполнении этих операций. Количество параллельных потоков будет кратно отношению времени Δt_{add} к Δt_{dat}. Например, при $\Delta t_{add}/\Delta t_{dat} = 2$ будет динамически сформировано два потока операций сложения (рис. 5).

Если темп поступления данных много выше времени выполнения операции сложения, то можно считать, что они успели поступить до начала суммирования, которое начнется практически одновременно для всех пар чисел. По завершении суммирования результаты также «мгновенно» поступают в асинхронный список, обеспечивая тем самым одновременное суммирование на втором слое. При этом общая схема вычислений будет соответствовать древовидной свертке, представленной на рис. 1.

Заключение

Использование асинхронных списков позволяет описать новый класс алгоритмов, параллелизм которых зависит от временных соотношений между выполняемыми основными и подготовительными операциями. Это ведет к специфическому подходу при построении параллельных алгоритмов и позволяет исследовать новые методы создания переносимых параллельных программ. В отличие от эквивалентных преобразований обобщенных функций, предлагаемых в [5] для адаптации параллельного алгоритма к конкретной вычислительной системе, использование асинхронных списков позволяет применять один и тот же алгоритм

Предложенный в работе механизм описания асинхронных вычислений предоставляет в распоряжение программиста дополнительные возможности. Его инструментальная поддержка на уровне языков обеспечивает использование новых методов разработки функционально-потоковых

параллельных программ, характеризующихся динамическим изменением параллелизма, в зависимости от временных соотношений между выполняемыми операциями. Это повышает гибкость разрабатываемых алгоритмов и позволяет использовать один и тот же алгоритм для динамического задания различных уровней параллелизма. Формируемые при этом программы позволяют описывать максимальный параллелизм решаемой задачи и не зависят от архитектур используемых вычислительных систем.

Литература

1. Хоар Ч. Взаимодействующие последовательные процессы / Пер. с англ. – М.: Мир, 1989. – 264 с.
2. Алгоритмы, математическое обеспечение и проектирование архитектур, многопроцессорных вычислительных систем / Под ред. А.П. Ершова. – М.: Наука, 1982. – 322 с.
3. Легалов А.И. Функциональный язык для создания архитектурно-независимых параллельных программ // Вычислительные технологии. − 2005. – Т.10, № 1. − С. 71–89.
4. Легалов А.И., Кузьмин Д.А., Казаков Ф.А., Привалихин Д.В. На пути к переносимым параллельным программам // Открытые системы. − 2003. – № 5. − С. 36–42.
5. Легалов А.И., Казаков Ф.А. Эквивалентные преобразования функционально-параллельных программ // Распределенные и кластерные вычисления: Избранные материалы Третьей школы-семинара / Институт вычислительного моделирования СО РАН. – Красноярск, 2004. – С. 134–141.