УДК 519.685.1

ЯЗЫК ПРОГРАММИРОВАНИЯ АСПЕКТ

С.Б. Арыков

Институт вычислительной математики и математической геофизики СО РАН, г. Новосибирск
E-mail: arykov@mail.ru

Рассматривается язык асинхронного параллельного программирования Аспект, позволяющий представлять алгоритмы с требуемой степенью непроцедурности. Описываются его ключевые особенности и синтаксис. На примере задачи умножения матриц продемонстрированы возможности языка Аспект по записи одного и того же алгоритма с различной степенью непроцедурности.

Ключевые слова:
асинхронные языки и системы программирования, автоматизация параллельной реализации численных моделей, язык Аспект, представление алгоритмов с различной степенью непроцедурности.

1. Введение

В ИВМиМГ СО РАН, г. Новосибирск, ведется разработка системы асинхронного параллельного программирования Аспект [1], предназначенной для реализации больших численных моделей, в которой за счет учета свойств численных алгоритмов, таких как, например, регулярность данных и вычислений, предполагается существенно повысить уровень языка программирования и обеспечить высокое качество конструируемых асинхронных программ.

Основная идея системы Аспект заключается в следующем: алгоритмы представляются с высокой степенью непроцедурности [2, 3], на основе которой для них автоматически генерируется представление меньшей степени непроцедурности, такой, что в системе программирования становится возможной генерация высокоэффективной параллельной программы для конкретного вычислителя.

Для представления алгоритмов с высокой степенью непроцедурности разработан язык программирования Аспект. В его основе лежит язык ОПАЛ [4], разработанный для описания задач на вычислительных моделях с массивами. В языке ОПАЛ собраны наиболее характерные способы непроцедурного задания массовых вычислений, однако этот язык предназначен, преимущественно, для теоретических работ и для практического применения требует конкретизации. Язык Аспект как раз является такой конкретизацией.

Одной из наиболее близких к настоящей являются работы по языку и системе программирования Поляр [5], однако там задача управления степенью непроцедурности представления алгоритмов не ставилась. Это универсальные язык и система программирования, в которых не учтены особенности структур численных алгоритмов. Из зарубежных работ необходимо отметить систему ALF [6] фирмы IBM.

2. Основные особенности языка Аспект

Наиболее важным отличием Аспект от ОПАЛ является то, что последний позволяет описать вычислительную модель, представляющую собой формализацию некоторой предметной области, тогда как Аспект позволяет задать конкретное представление алгоритма с необходимой степенью непроцедурности.

Основными особенностями языка являются:

1. Статическая типизация.
2. Явное описание информационных зависимостей между операциями.
3. Возможность повторного присваивания значения переменной.
4. Ориентация на регулярные структуры данных.
5. Отсутствие средств описания вычислений.

В императивных языках на множестве операций задается линейный порядок, что ограничивает возможности параллельного исполнения программ. В отличие от них, Аспект позволяет на множестве операций задать частичный порядок. Для этого все входные/выходные переменные каждой операции явно декларируются.

Как и в императивных языках, в Аспект допускается повторное присваивание значения переменной. Это означает, что в одной переменной программы могут находиться (в разное время) различные переменные алгоритма. Таким образом, Аспект является языком с частичным распределением ресурсов.

Система программирования Аспект ориентирована на решение задач численного моделирования, что нашло отражение и во входном языке: основой организации вычислений являются массовые операции [2], позволяющие эффективно обрабатывать регулярные структуры данных, а основной представления данных – массивы.

Аспект позволяет описать данные, операции, и взаимосвязи между ними. Для описания вычислений внутри операций (за исключением случая, когда операция реализуется модулем языка Аспект) предлагается использовать существующие языки программирования (в настоящее время поддерживаются лишь C/C++). Поэтому язык Аспект фактически является языком описания схемы программы.

3. Синтаксис языка

3.1. Общая структура программы

Аспект-программа представляет собой дерево модулей, вершина которого называется *главным модулем*.

Каждый модуль имеет имя, уникальное во всей программе, и может состоять из трех разделов: объявления переменных, объявления операций и описания управления:

```
program <ИмяМодуля> {
variables:
<Раздел объявления переменных>
operations:
<Раздел объявления операций>
control:
<Раздел описания управления>
}
```

В последнем разделе задается как потоковое, так и прямое управление. Таким образом, управление в языке Аспект полностью отделено от вычислений.

3.2. Раздел объявления переменных

Раздел объявления переменных состоит из трех подразделов: объявления входных переменных, объявления выходных переменных и объявления локальных переменных:

```
program <ИмяМодуля> {
in { <Объявление входных переменных> };
out { <Объявление выходных переменных> };
local { <Объявление локальных переменных> }
}
```

Подраздел **in** позволяет объявить переменные, являющиеся для данного модуля входными, а подраздел **out** – переменные, вычисляемые данным модулем в качестве результата. При этом память под переменные разделов **in** и **out** (за исключением главного модуля) не выделяется – предполагается, что она уже выделена в другом модуле, который передает/получает эти переменные. В подразделе **local** объявляются локальные переменные, которые будут уничтожены после завершения работы модуля.

В языке Аспект имеются следующие типы данных: базовые типы (целое – **int**, вещественное – **double**); структурные типы (массивы, записи, множества).

Поскольку язык Аспект ориентирован на решение задач численного моделирования, среди базовых отсутствуют такие распространенные типы, как логический, символьный, перечисление.

Объявление переменной производится одинаково в любом разделе:

```
<Тип> <Имя> {[<Индекс>]}
```

Если для переменной указан индекс, то она считается массивом.

3.3. Раздел объявления операций

В языке Аспект существуют три типа операций: простые, массовые и структурированные. Простая операция позволяет задать одну функцию и объявляется следующим образом:

```
<Имя>(in <П1>{,<П2>,...}; out <П1>{,<П2>, ...}) {
<Тело операции на внешнем языке >
}
```

где <Имя> – имя операции, после ключевого слова **in** следует список входных переменных, а после ключевого слова **out** – список выходных переменных. <Тело операции на внешнем языке> задает функцию, вычисляющую значения выходных переменных на основе значений входных переменных.

Массовая операция позволяет задать вычисление некоторой функции на области входных данных (области применимости). Ее объявление выглядит так:

```
<Имя>(in <П1>{,<П2>,...}; out <П1>{,<П2>,
...}) where <индекс>: <выр.1>...<выр.2>
{
<Тело операции на внешнем языке>
}
```

Семантика ключевых слов **in** и **out** аналогична объявлению простой операции. После ключевого слова **where** задается область применимости: указывается имя индекса и диапазон его изменения. Тело операции будет выполнено для каждого значения индекса.

Массовые операции могут быть редукционными. Это происходит в случае, когда у выходной переменной используется меньше индексов, чем у области применимости операции. Редукционная функция определяется телом операции.

Структурированные операции, как и простые, задают вычисление одной функции, однако в отличие от простых операций эта функция вычисляется другим модулем Аспект-программы. Структурированные операции объявляются так:

```
<Имя> (in <П1>{,<П2>,...}; out <П1>
{,<П2>, ...})
program <Имя> (<П1>=<П2>{,<П3>=<П4>,
...}) {
<Тело операции на внешнем языке>
}
```

Операции внутри модуля, реализующего структурированную операцию, могут обращаться только к собственным локальным переменным и к переменным, переданным в качестве параметров, и не могут обращаться к локальным переменным вызывающего модуля.

3.4. Раздел определения управления

Управление в языке Аспект задается определением частичного порядка на множестве операций. Если между операциями имеется информационная зависимость, управление является потоковым, иначе – прямым.

Для описания управления используется следующий синтаксис:

```
<Имя1> < <Имя2>,
```

где <Имя1> – имя операции, которая выполняется первой, <Имя2> – имя операции, которая выполняется после операции <Имя1>.

Все операции, для которых порядок не задан, могут выполняться одновременно.

3.5. Конфигурационный файл программы

На языке Аспект алгоритмы представляются с высокой степенью непроцедурности, однако, как отмечено в [1], высокая непроцедурность может препятствовать эффективному исполнению программы. Поэтому в системе Аспект предусмотрены специальные средства, позволяющие автоматически генерировать представление меньшей степени непроцедурности.

Алгоритмы генерации нового представления подробно изложены в [1]. Основная идея заключается в объединении нескольких экземпляров массовой операции в новую, более крупную операцию, называемую группой. Процесс группировки управляется конструкциями конфигурационного файла.

Конфигурационный файл состоит из трех секций: настройки системы Аспект, настройки группировки, настройки аппаратного обеспечения:

```
[Aspect]
<Объявление входных переменных>
[Fragments]
<Объявление выходных переменных>
[Hardware]
<Объявление локальных переменных>
```

В разделе **Aspect** задаются настройки транслятора (например, тип используемых массивов) и исполнительной подсистемы (например, способ измерения времени). В разделе **Fragments** задаются параметры группировки для каждой массовой операции. Раздел **Hardware** зарезервирован для будущих версий системы. В нем предполагается указывать настройки оборудования, под которое необходимо сгенерировать программу (например, топологию коммуникационной сети вычислителя). В текущей реализации поддержка этого раздела отсутствует.

Покажем, как работает группировка. Пусть в модуле Vector объявлена массовая операция *F*, преобразующая вектор *A* в вектор *B*:

```
F (in A[i]; out B[i]) where i: 0..m-1 {
<Код на внешнем языке, осуществляющий
преобразование>
}
```

Для того, чтобы сгруппировать экземпляры операции *F* в группы по 10, в конфигурационный файл необходимо внести следующую запись:

```
[Fragments]
program Vector
operation F
i 10
```

В этом случае, в результирующей программе вместо m *A*-блоков [1] будет $m/10$ *A*-блоков, каждый из которых будет содержать 10 экземпляров операции *F*. Сокращение количества *A*-блоков ведет к уменьшению накладных расходов на управление за счет сокращения количества единиц, которыми нужно управлять.

4. Пример использования языка Аспект. Алгоритм умножения матриц

Один и тот же алгоритм на языке программирования Аспект можно представить с различной степенью непроцедурности. Для иллюстрации этой возможности рассмотрим несколько представлений алгоритма умножения матриц.

Пусть имеются матрицы *A* и *B*. Тогда их произведение (матрица *C*) вычисляется по следующей формуле:

$$C_{ij} = \sum_k A_{ik} B_{kj}.$$

Фрагмент кода на языке C++, решающий поставленную задачу, приведен ниже:

```
for(int i=0; i<m; i++)
for(int j=0; j<l; j++)
for(int k=0; k<n; k++)
C[i][j] += A[i][k]*B[k][j];
```

С точки зрения параллельных вычислений этот код имеет существенный недостаток: он жестко задает последовательный порядок вычисления компонент матрицы *C*. В то же время исходный алгоритм не накладывал таких ограничений.

Язык Аспект позволяет объяснить системе, что компоненты матрицы *C* могут быть вычислены независимо. Пример такой программы приведен ниже.

```
program MultMatrix1 {
variables:
local {
double A[m][n], B[n][l], C[m][l]
}
operations:
F1(in A[i][0..n-1], B[0..n-1][j]; out
C[i][j]) where i: 0..m-1, j: 0..l-1 {
for(k=0; k<n; k++)
C[i][j] += A[i][k]*B[k][j];
}
}
```

Операция *F*1 задается на области, описываемой индексами *i* и *j*. Указывается, что для вычисления одного компонента *C*[*i*][*j*] необходим один вектор *A*[*i*][0..*n*–1] и один вектор *B*[0..*n*–1][*j*].

Степень непроцедурность программы «MultMatrix1» существенно выше степени непроцедурности фрагмента на языке C++ (все компоненты *C*[*i*][*j*] могут вычисляться параллельно). Тем не менее, в ней присутствует существенное прямое управление. В частности, суммирование по индексу *k*

выполняется целиком внутри операции *F*1, что приводит к невозможности разрезания матрицы по вертикали. Этот недостаток исправляет следующая программа «MultMatrix2».

```
program MultMatrix2 {
variables:
local {
double A[m][n], B[n][l], C[m][l]
}
operations:
F2(in A[i][k], B[k][j], C[i][j]; out
C[i][j]) where i: 0..m-1, j: 0..l-1,
k: 0..n-1 {
C[i][j] += A[i][k]*B[k][j];
}
}
```

В отличие от предыдущего примера, здесь каждый компонент *C*[*i*][*j*] вычисляется с помощью редукционной операции суммирования. Все операции суммирования с одинаковыми индексами *i* и *j* являются зависимыми, поскольку добавляют результат в одну и ту же переменную, и, следовательно, для них необходима синхронизация. В системе Аспект она выполняется автоматически.

Следует заметить, что хотя с теоретической точки зрения оба примера эквивалентны (на одинаковых данных вырабатывают одинаковый результат), на практике они могут отличаться из-за ошибок округления, возникающих вследствие представления чисел в ЭВМ конечным числом разрядов.

Непроцедурность представления алгоритма умножения матриц можно еще более увеличить. Для этого заметим, что в обоих предыдущих примерах мы последовательно производили умножение и сложение. Между тем, эти операции можно разделить. Третий пример демонстрирует, как это можно сделать в языке Аспект.

```
program MultMatrix3 {
variables:
local {
double A[m][n], B[n][l], C[m][l], D[n][l][m]
}
operations:
F31(in A[k][i], B[i][j]; out D[i][j][k])
where i: 0..n-1, j: 0..l-1, k: 0..m-1 {
D[i][j][k] = A[k][i]*B[i][j];
};
F32(in D[k][j][i], C[i][j]; out C[i][j])
where i: 0..m-1, j: 0..l-1, k: 0..n-1 {
C[i][j] += D[k][j][i];
}
control:
F31 < F32
}
```

В этом примере результаты всех умножений формируют трехмерный массив *D*. Итоговая матрица *C* получается поэлементным сложением всех слоев третьего измерения. Все операции F31 (операции умножения) являются независимыми и могут выполняться параллельно. Как только будет вычислен очередной компонент массива *D*, система Аспект может запустить на исполнение соответствующую операцию F32, добавляющую этот элемент в результирующую матрицу *C*.

5. Заключение

На практике к представлению алгоритма предъявляются противоречивые требования. С одной стороны, необходима высокая степень непроцедурности представления, т. к. она уменьшает зависимость программы от архитектуры и конфигурации оборудования конкретного вычислителя. С другой стороны, высокая степень непроцедурности влечет высокие затраты на организацию управления.

Система программирования Аспект в целом, и ее входной язык Аспект в частности, делают шаг на пути к преодолению указанного противоречия. Язык Аспект позволяет представлять алгоритмы с высокой степенью непроцедурности, а транслятор с него позволяет автоматически уменьшать степень непроцедурности представления алгоритма при генерации программы под конкретный вычислитель, причем программист имеет возможность изменять конечную степень непроцедурности в широких пределах.

СПИСОК ЛИТЕРАТУРЫ

1. Арыков С.Б., Малышкин В.Э. Система асинхронного параллельного программирования «Аспект» // Вычислительные методы и программирование. – 2008. – Т. 9. – № 1. – С. 205–209.
2. Вальковский В.А., Малышкин В.Э. Синтез параллельных программ и систем на вычислительных моделях. – Новосибирск: Наука, 1988. – 129 с.
3. Вальковский В.А., Малышкин В.Э. К уточнению понятия непроцедурности языков программирования // Кибернетика. – 1981. – № 3. – С. 55.
4. Малышкин В.Э. ОПАЛ – язык описания параллельных алгоритмов / В кн.: Теоретические вопросы параллельного программирования и многопроцессорные ЭВМ. – Новосибирск: ВЦ СО АН СССР, 1983. – С. 91–109.
5. Лельчук Т.И., Марчук А.Г. Язык программирования Поляр: описание, использование, реализация. – Новосибирск: ВЦ СО АН СССР, 1986. – 94 с.
6. Accelerated Library Framework for Cell Broadband Engine [Электронный ресурс]. – режим доступа: http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/41838EDB5A15CCCD002573530063D465. – 27.08.08.

Поступила 20.10.2008 г.