05.13.11

МЕТОДЫ ЗАДАНИЯ УПРАВЛЕНИЯ В ТЕХНОЛОГИИ ФРАГМЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

С.Б. Арыков

Институт вычислительной математики и математической геофизики СО РАН, г. Новосибирск
Новосибирский государственный технический университет, г. Новосибирск

Технология фрагментированного программирования ориентирована на разработку параллельных программ, решающих задачи численного моделирования. Конечная программа в этой технологии собирается из отдельных фрагментов вычислений, каждый из которых – независимая единица программы, содержащая входные/выходные переменные (фрагменты данных) и код их обработки (фрагмент кода). Для реализации этой технологии на вычислителях с общей памятью была разработана система параллельного программирования Аспект. Она включает в себя декларативный язык описания структуры фрагментированной программы (язык Аспект), транслятор с языка Аспект в C++, а также исполнительную подсистему. В статье представлены методы задания управления порядком исполнения фрагментов вычислений, разработанные и реализованные в системе программирования «Аспект». Все они основаны на использовании строгого частичного порядка на множестве фрагментов вычислений. Описаны различные типы управления: простое (между отдельными фрагментами), массовое (между множествами фрагментов), сложное (на основе логических операций; может задаваться как между отдельными фрагментами, так и между множествами фрагментов). Рассмотрены синтаксис и семантика основных конструкций языка Аспект, введенных для поддержки предложенных методов задания управления. Приведен пример решения задачи численного моделирования – задачи об *LU*-разложении матрицы – на основе которого продемонстрировано, как с помощью рассматриваемых методов задания управления можно представить численный алгоритм с высокой степенью непроцедурности. Предложенные подходы могут быть использованы как при реализации других инструментальных средств поддержки технологии фрагментированного программирования, так и встраиваться в качестве дополнительных способов задания управления (новой категории управляющих операторов) в современные языки программирования.

Ключевые слова: параллельное программирование, фрагментированное программирование, язык Аспект, методы управления параллельными вычислениями.

ВВЕДЕНИЕ

Стремительный прогресс в использовании параллелизма аппаратной частью привел к большому разрыву между предоставляемыми ресурсами и возможностями по их эффективному использованию с помощью имеющихся средств разработки. Специалистам, работающим в области численного моделирования, приходится все больше углубляться в архитектуру и нюансы работы современных вычислительных систем для получения приемлемого результата.

Для преодоления этой проблемы нужна технология, поддерживающая процесс разработки параллельных программ для задач численного моделирования. Важной особенностью этой технологии должно стать автоматизированное обеспечение параллельной программе ряда полезных свойств, таких, как должная степень непроцедурности [1], независимость алгоритма от оборудования (раздельное описание алгоритма и реализующей его программы), поддержка динамической балансировки нагрузки. Такая технология разрабатывается в Институте вычислительной математики и математической геофизики СО РАН. Она получила название «Технология фрагментированного программирования» [2].

Система программирования Аспект [3] является реализацией этой технологии для систем с общей памятью. Она включает в себя декларативный язык описания структуры фрагментированной программы и средства задания зависимостей между фрагментами этой структуры (язык Аспект), транслятор с языка Аспект в язык C++, а также исполнительную подсистему, которая обеспечивает выполнение описанных в языке Аспект фрагментов вычислений в соответствии с заданным порядком.

Теоретической основой проекта является специализированная асинхронная модель вычислений для систем с общей памятью [4]. При разработке системы Аспект учтены результаты проектов по созданию систем сборочного программирования [5], разработке параллельных программ для реализации больших численных моделей [6], а также исследования по нестандартным способам задания управления [7].

ЯЗЫК АСПЕКТ

Суть технологии фрагментированного программирования заключается в представлении алгоритма решения задачи в виде множества *фрагментов данных* и множества *фрагментов кода*. Фрагмент кода получает на вход набор входных фрагментов данных, на основе которых вычисляет набор выходных фрагментов данных. Подстановка фрагментов данных в качестве параметров фрагмента кода называется *применением* фрагмента кода к фрагментам данных (один и тот же фрагмент кода может применяться к различным фрагментам данных). Совокупность фрагмента кода и его входных и выходных фрагментов данных называется *фрагментом вычислений*. На множестве фраг-

ментов вычислений задаются ограничения на порядок их исполнения (управление).

Язык Аспект позволяет представлять алгоритмы во фрагментированной форме с высокой степенью непроцедурности. Его основное предназначение – описать «схему», по которой итоговая программа может быть собрана из фрагментов данных и вычислений, а также задать зависимости между этими фрагментами. Непосредственно вычисления на языке Аспект не описываются – для этого используются вставки на языке C++.

Язык Аспект ориентирован на задачи численного моделирования, поэтому акцент в нем сделан на регулярные структуры данных (многомерные массивы). В Аспект допускается повторное присваивание.

Программа на языке Аспект состоит из разделов (некоторые из них могут быть опущены), начинающихся с ключевых слов, выделенных ниже курсивом:

- *program* – задает имя программы (подпрограммы), а также определяет входные/выходные фрагменты данных для подпрограммы;
- *preface* – содержит объявления внешнего языка; здесь объявляются константы и типы данных, подключаются необходимые заголовочные файлы, настраивается препроцессор и выполняются другие действия, необходимые для корректной работы программы на C++;
- *data fragments* – объявление фрагментов данных; фрагменты данных конструируются на основе уже объявленных типов данных;
- *code fragments* – объявление фрагментов кода; указываются входные и выходные переменные фрагмента, а также код на языке C++, реализующий вычисления, связанные с фрагментом;
- *task data* – объявление данных задачи; структуры данных задачи конструируются из объявленных ранее фрагментов данных;
- *task computations* – объявление фрагментов вычислений; описывает применение фрагментов кода к фрагментам данных, в результате чего появляются простые либо массовые фрагменты вычислений;
- *task control* – задает управление на множестве фрагментов вычислений (см. подробней в следующем разделе).
- *end* – ключевое слово, завершающее любую программу (подпрограмму).

Детальное описание языка Аспект выходит за рамки статьи, однако смысл каждого раздела прямо соответствует понятиям фрагментированного программирования и интуитивно ясен из примеров Аспект-программ, приведенных ниже.

МЕТОДЫ ЗАДАНИЯ УПРАВЛЕНИЯ НА МНОЖЕСТВЕ ФРАГМЕНТОВ

Зависимости между фрагментами вычислений задаются в языке Аспект в декларативной форме в разделе *task control*. Каждая строка этого раздела задает управление с использованием следующего синтаксиса:

```
id1{[expression]} < id2{[expression]} {,
                    id3{[expression]}, …}}
```

где *id1* – имя фрагмента вычислений, который должен быть исполнен до того, как начнут свое исполнение фрагменты с именами *id2*, *id3* и так далее. Символ «<» (меньше) обозначает отношение строгого частичного порядка. В квадратных скобках могут быть указаны индексы и выражения (см. ниже раздел о массовом управлении). Строки отделяются друг от друга символом «;» (точка с запятой).

Для системы Аспект не имеет значения тип управления: потоковое (то есть определяемое информационными зависимостями между фрагментами вычислений), или прямое (определяемое другими соображениями, например необходимостью оптимизировать код или распределение ресурсов). В обоих случаях управление задается программистом и записывается синтаксически одинаково.

Отсутствие управления. Фрагменты вычислений, для которых явно не заданы зависимости, считаются независимыми. Покажем это на следующем примере. Пусть дана матрица M размера 1000×1000 и пусть каждый её элемент необходимо умножить на некоторое число Y. Эту задачу решает Аспект-программа, приведенная на рис. 1.

```
program Independent
preface {
   const int N = 500;
   const int K = 2;
   const int Y = 5.0;
};
data fragments
   double Matrix[N][N];
code fragments
   F(in Matrix A; out Matrix B) {
      for(int i=0; i<N; ++i)
         for(int j=0; j<N; ++j)
            B[i][j] = A[i][j]*Y;
   };
task data
   Matrix M[K][K];
task computations
   S[i][j]: F(M[i][j], M[i][j]) where i: 0..K-1, j: 0..K-1;
end
```

Рис. 1. Умножение матрицы на число

В разделе *data fragments* определяется новый фрагмент данных с именем *Matrix* и размером $N \times N$ элементов типа double. В разделе *code fragments* определяется один фрагмент кода с именем *F*, который получает на вход фрагмент данных типа *Matrix* и вычисляет в результате фрагмент данных типа *Matrix*. Вычисления задаются на языке C++. В разделе *task data* из фрагментов вычислений *Matrix* конструи-

руется исходная матрица *M*, так что в итоге в программе имеется $K \times K$ фрагментов данных. В заключении определяются фрагменты вычислений *S[i][j]*, для чего фрагмент кода *F* применяется к фрагментам данных $M_{ij}K \times K$ раз. Раздел *task control* в программе отсутствует. Это означает, что никаких ограничений на порядок исполнения фрагментов вычислений не накладывается и каждый фрагмент вычислений *S[i][j]* может выполняться независимо от других фрагментов, в том числе – параллельно с ними.

Простое управление. Зависимости, определенные между отдельными фрагментами вычислений, называются *простым управлением*. Для задания такого типа управления в разделе *task control* слева и справа от символа «<» необходимо указать точные имена фрагментов вычислений. Например, код

```
A < B;
B < C, D;
S[0][0] < S[1][1];
```

предписывает исполнить фрагмент вычислений *A* до начала исполнения фрагмента вычислений *B*. В свою очередь фрагмент вычислений *B* должен завершить исполнение до того, как могут начать исполняться фрагменты вычислений *C* и *D*, а фрагмент вычислений *S[0][0]* должен быть исполнен до начала исполнения фрагмента вычислений *S[1][1]*.

Массовое управление. Зависимости, определенные между группами фрагментов вычислений, называются *массовым управлением*. Для задания такого типа управления в разделе *task control* слева и справа от символа «<» необходимо указать группы фрагментов вычислений. Например, код

```
S1[i] < S2[i];
```

означает, что каждый фрагмент вычислений *S1[i]* должен выполниться перед соответствующим фрагментом вычислений *S2[i]*, где индекс *i* по умолчанию пробегает все возможные целочисленные значения пересечения областей значений индексов фрагментов вычислений *S1* и *S2*. Так, если определено четыре фрагмента вычислений *S1[0]*, *S1[1]*, *S1[2]*, *S1[3]* и четыре фрагмента вычислений *S2[1]*, *S2[2]*, *S2[3]*, *S2[4]*, то управление

```
S1[i] < S2[i];
```

эквивалентно управлению

```
S1[1] < S2[1];
S2[2] < S2[2];
S2[3] < S2[3];
```

Такое поведение можно изменить, если наложить на значения индекса некоторое условие. В этом случае управление будет задано только для тех значений индекса, для которых условие истинно. Например, с помощью записи

```
S1[i] <S2[i+1] where {i%2 == 0}
```

управление будет задано только для четных значений индекса *i*.

Помимо индексов, для задания массового управления можно использовать выражения вида <имя индекса> + <константа> и <имя индекса> – <константа>. Например,

```
S[p][q] < S[p][q+1];
S[p][q] < S[p+1][q];
```

означает, что каждый фрагмент вычислений *S[p][q]* зависит от своего левого и верхнего соседа, если расположить все *S[p][q]* на двумерной сетке.

Идентификаторы, используемые в качестве значений индексов при задании управления, никак не связаны с именами индексов соответствующих фрагментов вычислений. Например, фрагментпрограммы

```
task computations
   S1[i]: F1(A[i]) where i: 1..5;
   S2[i]: F2(A[i]) where i: 1..5;
task control
   S1[x] < S2[x];
```

является корректным. Здесь в качестве индекса при объявлении фрагментов вычислений использован идентификатор «*i*», а в качестве индекса при объявлении управления – идентификатор «*x*». Такая семантика обусловлена наличием сложного управления (см. ниже), когда в одной строке может быть указано множество фрагментов вычислений, у которых области применимости определены с использованием различных индексов.

Сложное управление. Зависимости, определенные с использованием логических операций, называются *сложным управлением*. Сложное управление может быть использовано совместно с простым и массовым. Для задания такого типа управления в разделе *task control* фрагменты вычислений слева от символа «<» объединяются с помощью логических операций «&» и «|», например:

```
(S1[i] & S2[i]) < S3[i];
(S1[i] | S2[i]) < S3[i];
((S2 & S3) | S4) < S5;
```

Операция «&» означает, что фрагмент вычислений справа от символа «<» (в примере выше – *S3[i]*) может начать исполнение, только если оба фрагмента вычислений, между которыми стоит символ «&» (в примере выше – *S1[i]*, *S2[i]*), завершили своё исполнение. Например, конструкция

```
(S1[i] & S2[i]) < S3[i];
```

задает ограничение, согласно которому *i*-й фрагмент вычислений *S3[i]* может быть выполнен только когда оба фрагмента вычислений *S1[i]* и *S2[i]* завершили свое исполнение. Управление действительно только для тех индексов, при которых одновременно определены все участвующие в нем фрагменты вычислений. Таким образом, семантика конструкций

```
(S1[i] & S2[i]) < S3[i];
```

и

```
S1[i] < S3[i];
S2[i] < S3[i];
```

совпадает только в том случае, если области значений индексов фрагментов вычислений *S1[i]*, *S2[i]* и *S3[i]* совпадают.

Операция «|» означает, что фрагмент вычислений справа от символа «<» (в примере выше – *S3[i]*) может начать исполнение, если завершил исполнение либо первый, либо второй фрагмент вычислений (в примере выше – *S1[i]*, *S2[i]*), либо они оба завершили исполнение. Управление действительно только для тех индексов, при которых одновременно определены все участвующие в нем фрагменты вычислений. Например, конструкция

(S1[i] | S2[i]) < S3[i];

задает ограничение, согласно которому *i*-й фрагмент вычислений *S3[i]* может быть выполнен, когда *S1[i]* или *S2[i]* (либо оба эти фрагмента) завершит свое исполнение.

РЕШЕНИЕ ПРИКЛАДНОЙ ЗАДАЧИ

Рассмотрим применение предложенных методов задания управления фрагментами вычислений для решения задачи разложения квадратной матрицы A в произведение матриц L и U (LU-разложения).

Пусть имеется матрица $A = (a_{ij})$, $i = 1…N$, $j = 1…N$. Её разложение в виде $A = LU$, где $L = (l_{ij})$ – нижняя треугольная матрица, а $U = (u_{ij})$ – верхняя треугольная матрица с единицами на главной диагонали, может быть вычислено по формулам [8]:

$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj} \quad (i \geq j)$$

$$u_{ij} = \frac{1}{l_{ii}}\left(a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj}\right) \quad (i<j)$$

Подробное описание фрагментации алгоритма и результаты измерений производительности приведены в работе [9]. Для иллюстрации методов задания управления в языке Аспект интерес представляет текст программы LU-разложения, приведенный на рис. 2.

```
program LUDecomposition
preface {
  const int m = 90;
  const int n = 56;
};
data fragments
  double Matrix[m][m];
code fragments
  F1(in Matrix A; out Matrix B) {
    for(int i=0; i<m; ++i)
      for(int j=i+1; j<m; ++j) {
        B[j][i] = A[j][i]/A[i][i];
        for(int k=i+1; k<m; ++k)
          B[j][k] = A[j][k] – A[j][i]*A[i][k];
      }
  };
  F2(in Matrix A, Matrix B; out Matrix C) {
    for(int i=0; i<m; ++i)
      for(int j=i+1; j<m; ++j)
        for(int k=0; k<m; ++k)
          C[j][k] = B[j][k] - A[j][i]*B[i][k];
  };
  F3(in Matrix A, Matrix B; out Matrix C) {
    for(int i=0; i<m; ++i )
      for(int j=0; j<m; ++j) {
        C[i][j] = B[i][j] / A[j][j];
        for(int k=j+1; k<m; ++k)
          C[i][k] = B[i][k] - B[i][j]*A[j][k];
      }
  };
  F4(in Matrix A, Matrix B, Matrix C; out Matrix D)
  {
    for(int i=0; i<m; ++i)
      for(int j=0; j<m; ++j) {
        double sum = 0;
        for(int k=0; k<m; ++k)
          sum += A[i][k]*B[k][j];
        D[i][j] = C[i][j] - sum;
      }
  };
task data
  Matrix A[n][n];
task computations
  G1[i]: F1(A[i][i], A[i][i])
where i: 0..n-1 priority 0;
  G2[i][j]: F2(A[i][i], A[i][j], A[i][j])
where i: 0..n-2, j: i+1..n-1;
  G3[i][j]: F3(A[j][j], A[i][j], A[i][j])
where i: j+1..n-1, j: 0..n-2;
  G4[k][i][j]: F4(A[i][k], A[k][j], A[i][j], A[i][j])
where k: 0..n-1, i: k+1..n-1, j: k+1..n-1;
task control
  G1[i] < G2[i][], G3[][i];
  G2[i][j] < G4[i][][j];
  G3[i][j] < G4[j][i][];
  G4[i-1][i][i] < G1[i];
  G4[i-1][i][j] < G2[i][j];
  G4[j-1][i][j] < G3[i][j];
  G4[k][i][j] < G4[k+1][i][j];
end
```

Рис. 2. Программа LUDecomposition

Результатыизмерений производительности программы LUDecomposition насистеме HP ProLiant DL580 G5 (4 процессора Intel Xeon X7350, 16 ядер, 256 Гбайт RAM) представленывтабл. 1.

Табл. 1. Результаты тестирования программы LUDecomposition

Характеристика	Количество ядер				
	1	2	4	8	16
Время счета, с	31,01	15,6	7,93	4,13	2,35
Ускорение	1	1,99	3,91	7,51	13,2

Программа LUDecomposition представляет алгоритм LU-разложения с высокой степенью непроцедурности: практически все фрагменты вычислений могут стартовать сразу, как только для них будут вычислены входные данные, что обеспечивает хорошие показатели ускорения.

БЛИЗКИЕ РАБОТЫ

Конструирование управления в языке Аспект похоже на подход, предложенный в работе [7] и реализованный в языке Барс [10]. Программа на языке Барс задаёт выполнение действия. Действие – это либо модуль, либо оператор. Каждый модуль имеет свою локальную память, своё локальное управление и свой набор действий, и, таким образом, служит средством иерархической структуризации действий. По умолчанию все действия выполняются независимо и асинхронно. Любое управление необходимо задать явно, что можно сделать статическим или динамическим способом. Статическое управление задаётся управляющими выражениями (С-формулами) с помощью С-языка и определяет статические связи между действиями. Семантика этих выражений описывается в терминах регулярных сетей Петри [11]. Динамическое управление задаётся с помощью логического выражения, так что действие может быть выполнено, только если выражение имеет значение «истина».

Главное отличие предложенного метода состоит в явной поддержке языком Аспект массового управления. Отличается и способ задания управления: в работе [7] используются регулярные сети Петри, в языке Барс – механизм «просачивания» [10], а в языке Аспект – строгий частичный порядок [4].

Другой попыткой реализовать фрагментированную технологию программирования является система автоматизированного конструирования численных параллельных программ LuNA [12]. В ней процесс вычислений рассматривается как двудольный ориентированный граф, в котором одна доля – фрагменты единственного присваивания (представляют данные задачи), а другая доля – фрагменты вычислений без побочных эффектов (представляют вычисления задачи). Управление задается дугами графа, которые определяют множества входных и выходных фрагментов данных для каждого фрагмента вычислений. При этом для некоторых классов алгоритмов LuNA может генерировать статическое управление, сокращая возможные варианты реализации вплоть до императивного исполнения.

Среди зарубежных исследований наиболее интересен проект PLASMA [13] под руководством Джека Донгарры. В проекте реализуется идея представления алгоритмов линейной алгебры в виде множества фрагментов, или, в терминологии авторов, «плиток» (tiles), что близко к представлению в виде множества фрагментов вычислений. Однако реализуется она в форме библиотеки, то есть конкретной реализации конкретных алгоритмов.

ЗАКЛЮЧЕНИЕ

Предложенные методы задания управления позволяют представлять фрагментированные алгоритмы с высокой степенью непроцедурности и обеспечивают полное отделение управления от вычислений. На базе этих методов разработана и реализована система программирования Аспект, позволяющая использовать технологию фрагментированного программирования на системах с общей памятью.

Предложенные методы могут применяться при создании альтернативных реализаций технологии фрагментированного программирования, новых языков и систем параллельного программирования, а также могут быть встроены в качестве дополнительных средств управления в существующие языки программирования.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Малышкин, В.Э. Параллельное программирование мультикомпьютеров [Текст] / В.Э. Малышкин, В.Д. Корнеев. – Новосибирск: Изд-во НГТУ, 2006. – 296 с.

2. Малышкин, В.Э. Технология фрагментированного программирования [Текст] / В.Э. Малышкин // Вестник ЮУрГУ. Серия «Вычислительная математика и информатика». – 2012. – № 46. – С.45-55.

3. Arykov, S.B., andMalyshkin, V.E. AsynchronousLanguageand-SystemofNumericalAlgorithmsFragmentedProgramming, LNCS, vol. 5698, pp. 1–7, 2009.

4. Арыков, С.Б. Асинхронная модель вычислений над общей памятью [Текст] / С.Б. Арыков // Вестник УГАТУ. – 2016. – Т. 20. – № 3. – С. 114-121.

5. Kraeva, M.A., and Malyshkin, V.E. Assembly Technology for Parallel Realization of Numerical Models on MIMD-Multicomputers, International Journal on Future Generation Computer Systems, vol. 17, no. 6, pp. 755–765, 2001.

6. Киреев, С.Е. Параллельная реализация метода частиц в ячейках для моделирования задач гравитационной космодинамики [Текст] / С.Е. Киреев // Автометрия. – 2006. – № 3. – С. 32–39.

7. Котов, В.Е. Параллельное программирование с типами управления [Текст] / В.Е. Котов // Кибернетика. – 1979. – № 3. – С. 1–13.

8. Фаддеев, Д.К. Вычислительные методы линейной алгебры [Текст] / Д.К. Фаддеев, В.Н. Фаддеева. – М.: Наука, 1963. – 2-е изд. – 656 с.

9. Арыков, С.Б. Решение прикладных задач в системе параллельного программирования Аспект [Текст] / С.Б. Арыков // Вестник ТГУ. Управление, вычислительная техника и информатика. – 2018. – № 45. – С. 59–67.

10. Бульонков, М.А. Базовый язык параллельного программирования Барс [Текст] / М.А. Бульонков, А.В. Быстров, Н.Н. Дудоров, В.Е. Котов // Программирование. – 1986. – № 6. – С. 32–40.

11. Котов, В.Е. Алгебра регулярных сетей Петри [Текст] / В.Е. Котов // Кибернетика. – 1980. – № 5. – С. 10–18.

12. Malyshkin, V.E., and Perepelkin, V.A. Optimization methods of parallel execution of numerical programs in the Luna fragmented programming system, The Journal of Supercomputing. Special issue on Enabling Technologies for Programming Extreme Scale Systems, vol. 61, no. 1, pp. 235-248, 2012.

13. Buttari, A., Langou, J., Kurzak, J., and Dongarra, J. A class of parallel tiled linear algebra algorithms for multicore architectures, Parallel Computing, vol. 35, no. 1, pp. 38–53, 2009.

Арыков Сергей Борисович, к.ф.-м.н., доцент кафедры Параллельных вычислительных технологий Новосибирского государственного технического университета, младший научный сотрудник Института вычислительной математики и математической геофизики СО РАН, тел. (383) 330-89-94,e-mail: arykov@sscc.ru

METHODS OF DEFINING ORDER OF EXECUTION IN FRAGMENTED PROGRAMMING TECHNOLOGY

S.B. Arykov

Institute of Computational Mathematics and Mathematical Geophysics Siberian Branch of the Russian Academy of Sciences, Novosibirsk

Novosibirsk State Technical University, Novosibirsk

The technology of fragmented programming is focused on the development of paral-lel programs that solve numerical problems. The final program in this technology is assem-bled from separate computation fragments, each of which is an independent program unit containing input and output variables (data fragments) and the code of their processing (code fragment). To implement this technology on computers with shared memory, a paral-lel programming system Aspect was developed. It includes a declarative language for de-scribing the structure of a fragmented program (the Aspect language), a translator from the Aspect language to C ++ and an executive subsystem. The article presents the methods of defining order of execution of computation fragments, developed and implemented in the programming system "Aspect". All of them are based on the use of strict partial order on a set of computation fragments. Various approaches to define order of execution are de-scribed: simple (between separate fragments), massive (between sets of fragments), com-plex (based on logical operations; can be set between separate fragments, and between sets of fragments). The syntax and semantics of the main structures of the Aspect language, in-troduced to support the proposed methods of the control, are considered.An example of solving LU decomposition of a matrix numerical problem is given. Based on that example it was demonstrated how using methods offered one can present a numerical algorithm with a high degree of nonprocedurality. The proposed approaches can be used to implement other tools to support the technology of fragmented programming or can be integrated as additional ways to set the control (new category of control operators) into modern programming languages.

Index terms: parallel programming, technology of fragmented programming, asynchronous model of computing, programming system Aspect, language Aspect, methods for controlling parallel computations.

REFERENCES

1. Malyshkin, V.E., Korneev, V.D. Parallel programming multicomputers, Novosibirsk, NSTU Publ., 2006, 296 p.

2. Malyshkin, V.E. Fragmented Programming Technology, Bulletin of the South Ural State University. Series “Computational mathematics and software engineering”, 2012, no. 46, pp. 45–55.

3. Arykov, S.B., Malyshkin, V.E. Asynchronous Language and System of Numerical Algorithms Fragmented Programming, LNCS, 2009, vol. 5698, pp. 1–7.

4. Arykov, S.B. Asynchronous model of computation on shared memory, Bulletin of the Ufa State Aviation Technical University, 2016, vol. 20, no. 3, pp. 114-121.

5. Kraeva, M.A., Malyshkin, V.E. Assembly Technology for Parallel Realization of Numerical Models on MIMD-Multicomputers, International Journal on Future Generation Computer Systems, 2001, vol. 17, no. 6, pp. 755–765.

6. Kireev, S.E. Parallel implementation of the particle-in-cells method for modeling of gravitational cosmodynamics, Optoelectronics, Instrumentation and Data Processing, 2006, no. 3, pp. 32–39.

7. Kotov, V.E. Parallel programming with control types, Cybernetics, 1979, no. 3, pp. 1–13.

8. Faddeev, D.K., Faddeeva, V.N. Computational methods of linear algebra, Moscow, Nauka Publ., 1963, 656 p.

9. Arykov, S.B. Solution of applied problems in the parallel programming system Aspect, Tomsk State University Journal of Control and Computer Science, 2018, no. 45, pp. 59–67.

10. Buljonkov, M.A., Bystrov, A.V., Dudorov, N.N., Kotov, V.E. The basic language of parallel programming Bars, Programming and Computer Software, 1986, no. 6, pp. 32–40.

11. Kotov, V.E. Algebra of regular Petri nets, Cybernetics, 1980, no. 5, pp. 10–18.

12. Malyshkin, V.E., Perepelkin, V.A. Optimization methods of parallel execution of numerical programs in the Luna fragmented programming system. The Journal of Supercomputing. Special issue on Enabling Technologies for Programming Extreme Scale Systems, 2012, vol. 61, no. 1, pp. 235-248.

13. Buttari, A., Langou, J., Kurzak, J., Dongarra, J. A class of parallel tiled linear algebra algorithms for multicore architectures, Parallel Computing, 2009, vol. 35, no. 1, pp. 38–53.

Arykov Sergey Borisovich, PhD (Physics and Mathematics), associate professor at the department of parallel computing technologies, Novosibirsk State Technical University; junior researcher at Institute of Computational Mathematics and Mathematical Geophysics of Siberian Branch of Russian Academy of Science, (383) 330-89-94, e-mail: arykov@sscc.ru