

Реализация алгоритмов автоматической генерации фрагментов вычислений и данных по их высокоуровневому описанию

Синюков Валерий Константинович

Новосибирский государственный университет

Летняя XLIII школа-конференция по параллельному программированию

Научный руководитель: Перепёлкин Владислав Александрович, к.т.н., ст. преп. каф. ПВ ФИТ

12 июля 2024 года

Введение

Для разработки параллельных программ специалистам приходится решать **сложные**, напрямую не связанные с реализуемым алгоритмом, задачи:

- Декомпозиция данных
- Обеспечение коммуникаций между процессами и потоками
- Синхронизация потоков и процессов
- Балансировка нагрузки по потокам и процессам

Параллельные программы **сложны** в отладке.

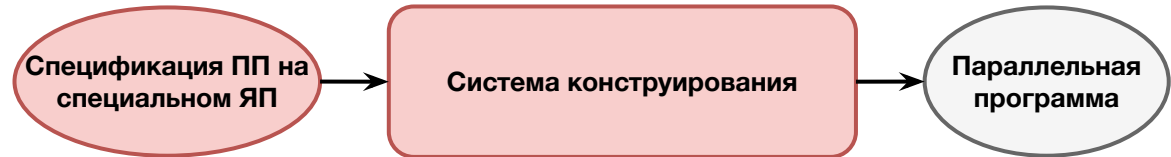
Таким образом, специалистам приходится выполнять **трудоемкую работу**, которая находится **вне области их специализации**.

Одним из подходов к решению этой проблемы является **автоматическое конструирование параллельных программ**.

Подходы к автоматическому конструированию параллельных программ

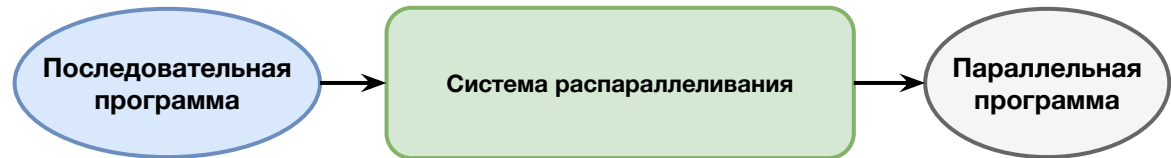
Системы, которые автоматически конструируют параллельные программы (ПП), исходя из заданной спецификации на специальном языке программирования (ЯП):

- HORMA
- DVM
- LuNA



Системы автоматического распараллеливания последовательных программ:

- Par4All
- PLUTO
- AutoPar



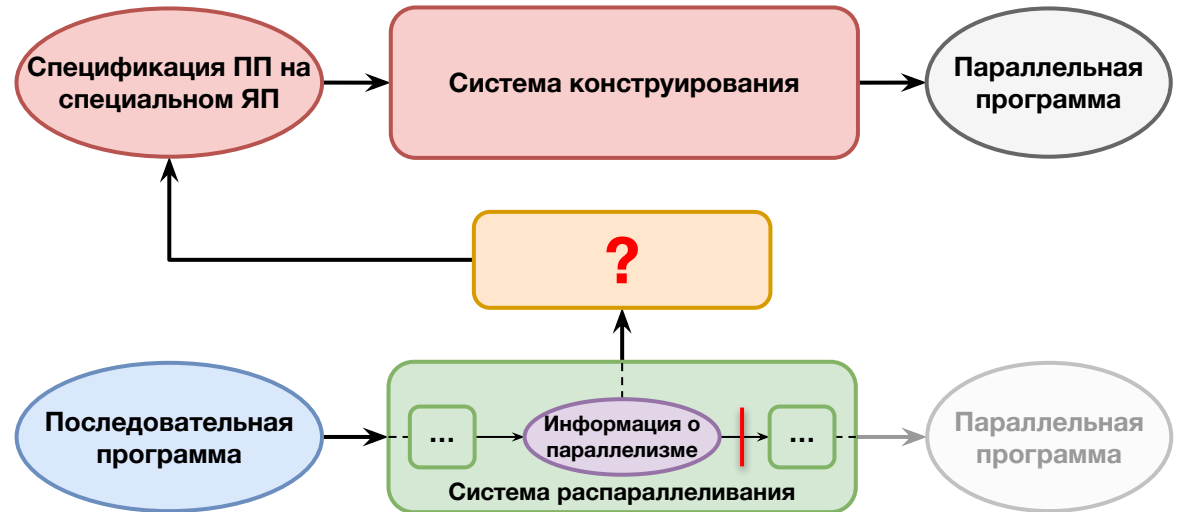
Такие системы далее будут называться **системами конструирования** и **распараллеливания** соответственно.

Связывание систем автоматического распараллеливания и автоматического конструирования

Процесс работы системы распараллеливания можно условно разделить на два этапа.

1. Извлечение из программы **информации о параллелизме**, т. е. информации о ее независимых частях.

2. Использование этой информации для **перестроения программы** таким образом, чтобы эти независимые части можно было выполнять **параллельно**.

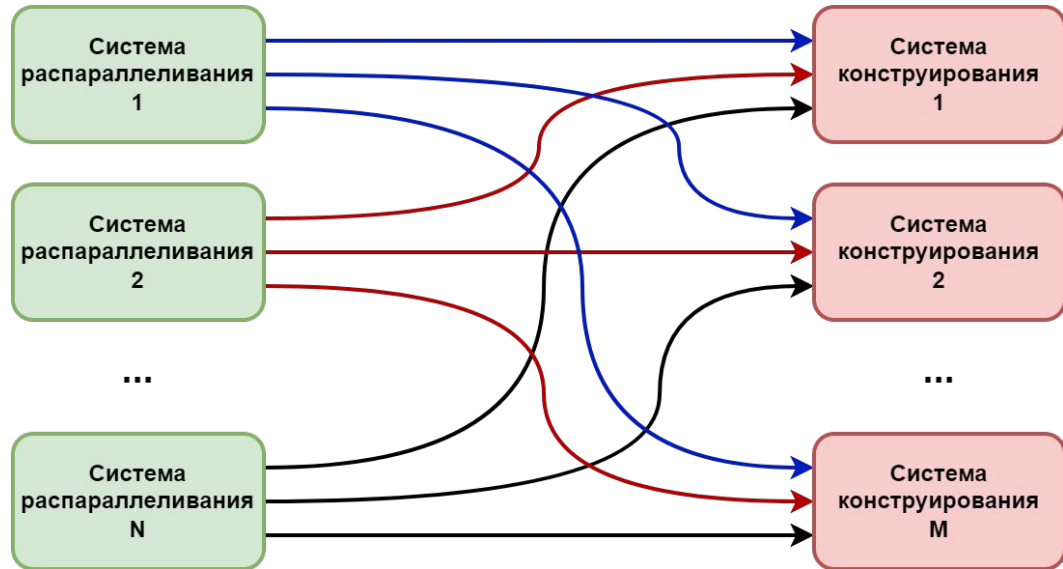


Информацию о параллелизме предлагается использовать для **автоматического создания спецификации** для системы конструирования.

Связывание систем автоматического распараллеливания и автоматического конструирования. Актуальность

У каждой системы автоматического конструирования параллельных программ есть **своя область применения**.

Обеспечение возможности автоматического создания спецификаций для систем конструирования **расширит множество приложений**, для которых возможно автоматическое конструирование эффективных параллельных программ.



Несовместимость систем распараллеливания и систем конструирования

Системы распараллеливания и системы конструирования в общем случае работают в **разных моделях вычислений.**

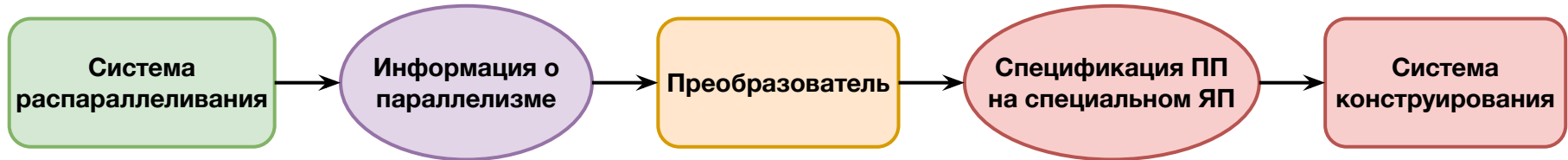


Несовместимость систем распараллеливания и систем конструирования

Системы распараллеливания и системы конструирования в общем случае работают в **разных моделях вычислений**.

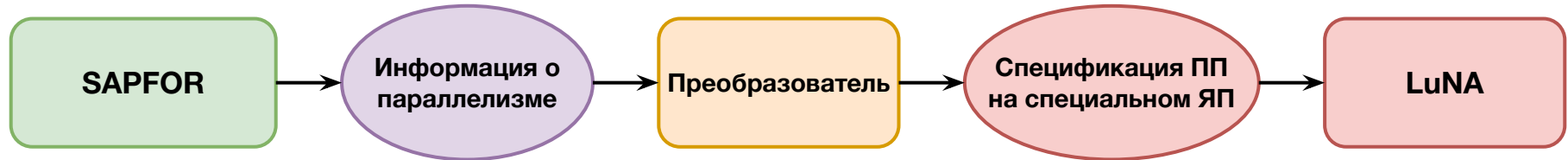
Информацию о параллелизме, выявленную системой распараллеливания, нужно **дополнить** и **преобразовать**, чтобы система конструирования смогла по ней составить параллельную программу.

Для выполнения этой задачи требуется разработать систему, которая далее будет называться **преобразователем**.



Цель объемлющей работы

Связывание систем SAPFOR и LuNA для автоматического распараллеливания последовательных программ для класса задач на примере итерационного метода Якоби решения систем линейных уравнений.



Система конструирования LuNA

LuNA — **L**anguage for **N**umerical **A**lgorithms

Система LuNA по описанию задачи в терминах предметной области строит параллельную программу.

Работает в модели вычислений фрагментированного программирования.

Разрабатывается в ИВМиМГ СО РАН.

<https://ssd.sccc.ru/luna>

Victor E. Malyshkin, Vladislav A. Perepelkin. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem // Parallel Computing Technologies. 11th International Conference, PaCT 2011, Proceedings. LNCS 6873. Springer, 2011. pp. 53-61.

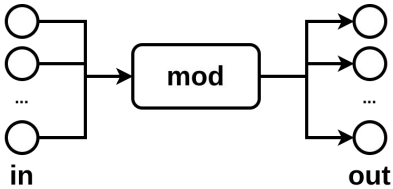
LuNA-программа

LuNA-программа задает спецификацию требуемой параллельной программы.

Данные представлены с помощью переменных **единственного** присваивания, которые называются *фрагментами данных*.

Вычисления представлены с помощью операций, которые называются *фрагментами вычислений*.

Фрагмент вычислений — это триплет вида

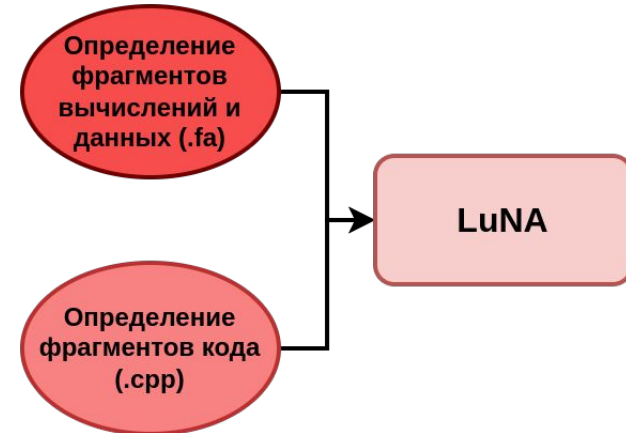


Где

- **in** — множество входных фрагментов данных.
- **out** — множество выходных фрагментов данных.
- **mod** — модуль, который из входных фрагментов данных вычисляет выходные. В LuNA модуль называется *фрагментом кода*.

Фрагменты данных и фрагменты вычислений описываются на языке программирования LuNA.

Фрагменты кода описываются на языке программирования C++.



LuNA-программа

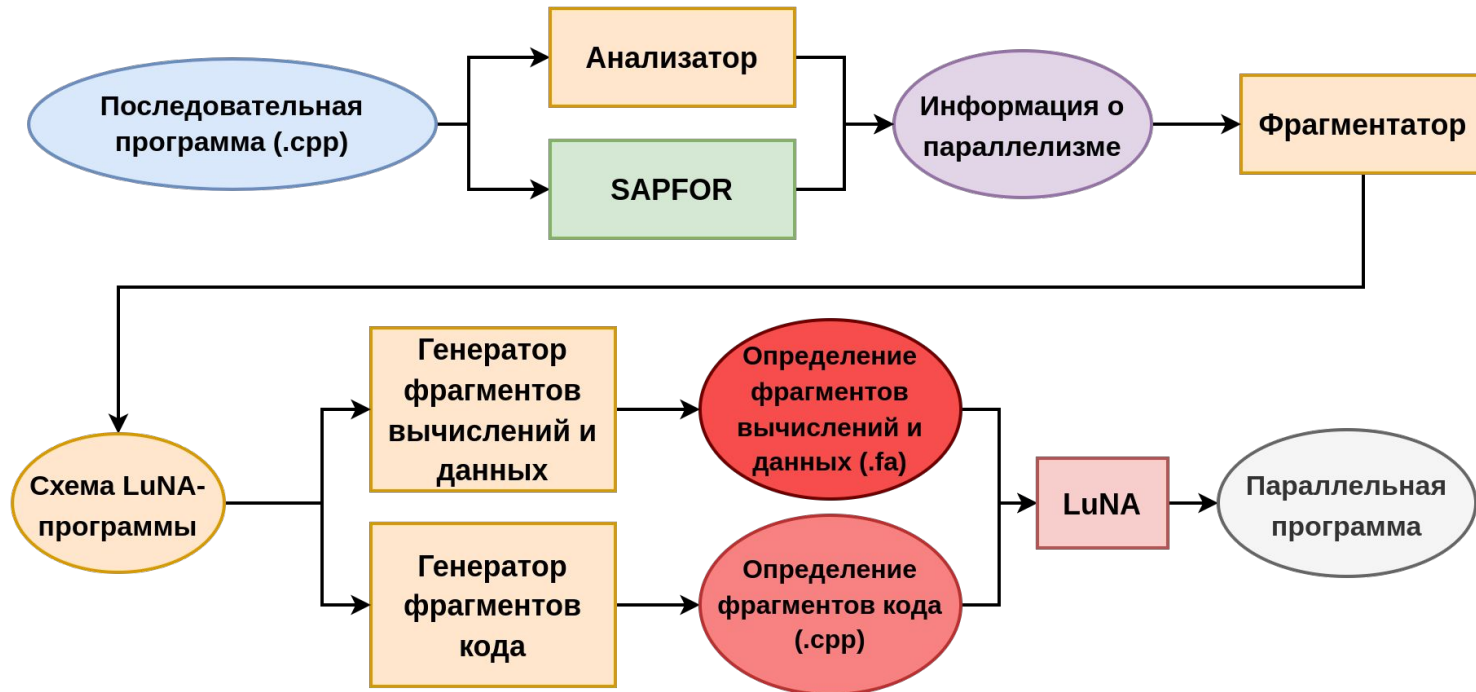
Для описания потенциально бесконечного множества фрагментов вычислений и данных в LuNA присутствуют **массовые** и **условный** операторы: `for`, `while`, `if`.

Имена фрагментов данных строятся из **базовой** и **индексной** частей.

`x` `[1]` `[y]` `[z [i]]`

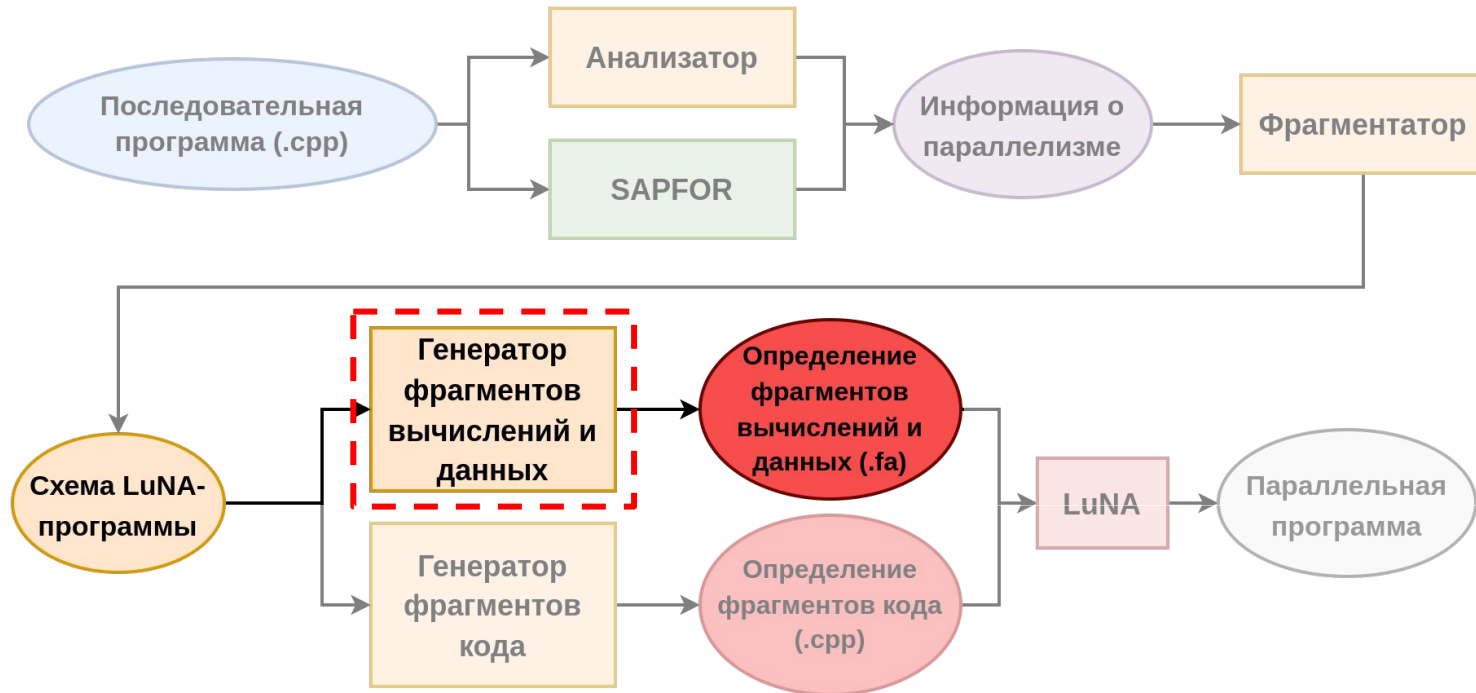
базовая часть | индексная часть

Предлагаемый подход к автоматическому распараллеливанию последовательных программ на основе систем SAPFOR и LuNA



Цель на школу

Реализация алгоритмов генератора фрагментов вычислений и данных.



Метод Якоби. Исходная программа

```

01:  #define L 384
02:  #define ITMAX 100
03:  ...
04:  double B[L][L][L];
05:  double A[L][L][L];
06:
07:  int main(int an, char **as)
08:  {
09:      int i, j, k, it;
10:      double eps, MAXEPS = 0.5f;
11:      for (i = 0; i < L; i++)
12:          for (j = 0; j < L; j++)
13:              for (k = 0; k < L; k++)
14:                  {
15:                      A[i][j][k] = 0;
16:                      if (i == 0 || j == 0
17:                          || k == 0 || i == L - 1
18:                          || j == L - 1 || k == L - 1)
19:                          B[i][j][k] = 0;
20:                      else
21:                          B[i][j][k] = 4 + i + j + k;
22:                  }
23:      for (it = 1; it <= ITMAX; it++)
24:      {
25:          eps = 0;

```

```

26:          for (i = 1; i < L - 1; i++)
27:              for (j = 1; j < L - 1; j++)
28:                  for (k = 1; k < L - 1; k++)
29:                      {
30:                          double tmp = fabs(B[i][j][k] - A[i][j][k]);
31:                          eps = Max(tmp, eps);
32:                          A[i][j][k] = B[i][j][k];
33:                      }
34:
35:          for (i = 1; i < L - 1; i++)
36:              for (j = 1; j < L - 1; j++)
37:                  for (k = 1; k < L - 1; k++)
38:                      B[i][j][k] = (A[i - 1][j][k] + A[i][j - 1][k]
39:                                  + A[i][j][k - 1] + A[i][j][k + 1]
40:                                  + A[i][j + 1][k] + A[i + 1][j][k]) / 6.0;
41:
42:          printf(" IT = %4i   EPS = %14.7E\n", it, eps);
43:          if (eps < MAXEPS)
44:              break;
45:      }
46:      ... // вывод результатов работы программы
47:      return 0;
48:  }
49:
50:

```

Метод Якоби. Исходная программа

Гнездо циклов инициализации массивов A и B

```

01:  #define L 384
02:  #define ITMAX 100
03:  ...
04:  double B[L][L][L];
05:  double A[L][L][L];
06:
07:  int main(int an, char **as)
08:  {
09:      int i, j, k, it;
10:      double eps, MAXEPS = 0.5f;
11:      for (i = 0; i < L; i++)
12:          for (j = 0; j < L; j++)
13:              for (k = 0; k < L; k++)
14:                  {
15:                      A[i][j][k] = 0;
16:                      if (i == 0 || j == 0
17:                          || k == 0 || i == L - 1
18:                          || j == L - 1 || k == L - 1)
19:                          B[i][j][k] = 0;
20:                      else
21:                          B[i][j][k] = 4 + i + j + k;
22:                  }
23:      for (it = 1; it <= ITMAX; it++)
24:      {
25:          eps = 0;

```

```

26:          for (i = 1; i < L - 1; i++)
27:              for (j = 1; j < L - 1; j++)
28:                  for (k = 1; k < L - 1; k++)
29:                      {
30:                          double tmp = fabs(B[i][j][k] - A[i][j][k]);
31:                          eps = Max(tmp, eps);
32:                          A[i][j][k] = B[i][j][k];
33:                      }
34:
35:          for (i = 1; i < L - 1; i++)
36:              for (j = 1; j < L - 1; j++)
37:                  for (k = 1; k < L - 1; k++)
38:                      B[i][j][k] = (A[i - 1][j][k] + A[i][j - 1][k]
39:                                  + A[i][j][k - 1] + A[i][j][k + 1]
40:                                  + A[i][j + 1][k] + A[i + 1][j][k]) / 6.0;
41:
42:          printf(" IT = %4i   EPS = %14.7E\n", it, eps);
43:          if (eps < MAXEPS)
44:              break;
45:      }
46:      ... // вывод результатов работы программы
47:      return 0;
48:  }
49:
50:

```

Метод Якоби. Исходная программа Цикл с основной вычислительной нагрузкой

```

01:  #define L 384
02:  #define ITMAX 100
03:  ...
04:  double B[L][L][L];
05:  double A[L][L][L];
06:
07:  int main(int an, char **as)
08:  {
09:      int i, j, k, it;
10:      double eps, MAXEPS = 0.5f;
11:      for (i = 0; i < L; i++)
12:          for (j = 0; j < L; j++)
13:              for (k = 0; k < L; k++)
14:                  {
15:                      A[i][j][k] = 0;
16:                      if (i == 0 || j == 0
17:                          || k == 0 || i == L - 1
18:                          || j == L - 1 || k == L - 1)
19:                          B[i][j][k] = 0;
20:                      else
21:                          B[i][j][k] = 4 + i + j + k;
22:                  }
23:      for (it = 1; it <= ITMAX; it++)
24:      { ←
25:          eps = 0;

```

```

26:          for (i = 1; i < L - 1; i++)
27:              for (j = 1; j < L - 1; j++)
28:                  for (k = 1; k < L - 1; k++)
29:                      {
30:                          double tmp = fabs(B[i][j][k] - A[i][j][k]);
31:                          eps = Max(tmp, eps);
32:                          A[i][j][k] = B[i][j][k];
33:                      }
34:
35:          for (i = 1; i < L - 1; i++)
36:              for (j = 1; j < L - 1; j++)
37:                  for (k = 1; k < L - 1; k++)
38:                      B[i][j][k] = (A[i - 1][j][k] + A[i][j - 1][k]
39:                                  + A[i][j][k - 1] + A[i][j][k + 1]
40:                                  + A[i][j + 1][k] + A[i + 1][j][k]) / 6.0;
41:
42:          printf(" IT = %4i  EPS = %14.7E\n", it, eps);
43:          if (eps < MAXEPS)
44:              break;
45:      } ←
46:      ... // вывод результатов работы программы
47:      return 0;
48:  }
49:
50:

```

Метод Якоби. Исходная программа

Гнезда циклов вычислений A и B

```

01:  #define L 384
02:  #define ITMAX 100
03:
04:  double B[L][L][L];
05:  double A[L][L][L];
06:
07:  int main(int an, char **as)
08:  {
09:      int i, j, k, it;
10:      double eps, MAXEPS = 0.5f;
11:      for (i = 0; i < L; i++)
12:          for (j = 0; j < L; j++)
13:              for (k = 0; k < L; k++)
14:                  {
15:                      A[i][j][k] = 0;
16:                      if (i == 0 || j == 0
17:                          || k == 0 || i == L - 1
18:                          || j == L - 1 || k == L - 1)
19:                          B[i][j][k] = 0;
20:                      else
21:                          B[i][j][k] = 4 + i + j + k;
22:                  }
23:      for (it = 1; it <= ITMAX; it++)
24:      {
25:          eps = 0;

```

```

26:          for (i = 1; i < L - 1; i++)
27:              for (j = 1; j < L - 1; j++)
28:                  for (k = 1; k < L - 1; k++)
29:                      {
30:                          double tmp = fabs(B[i][j][k] - A[i][j][k]);
31:                          eps = Max(tmp, eps);
32:                          A[i][j][k] = B[i][j][k];
33:                      }
34:
35:          for (i = 1; i < L - 1; i++)
36:              for (j = 1; j < L - 1; j++)
37:                  for (k = 1; k < L - 1; k++)
38:                      B[i][j][k] = (A[i - 1][j][k] + A[i][j - 1][k]
39:                                  + A[i][j][k - 1] + A[i][j][k + 1]
40:                                  + A[i][j + 1][k] + A[i + 1][j][k]) / 6.0;
41:
42:          printf(" IT = %4i   EPS = %14.7E\n", it, eps);
43:          if (eps < MAXEPS)
44:              break;
45:      }
46:      ... // вывод результатов работы программы
47:      return 0;
48:  }
49:
50:

```

Фрагментация по пространству — параллельное выполнение циклов

Составляется разбиение множества итераций **вычислительноемких гнезд циклов без зависимостей**.

Предлагается выполнять каждое множество из этого разбиения в **отдельном фрагменте вычислений**.

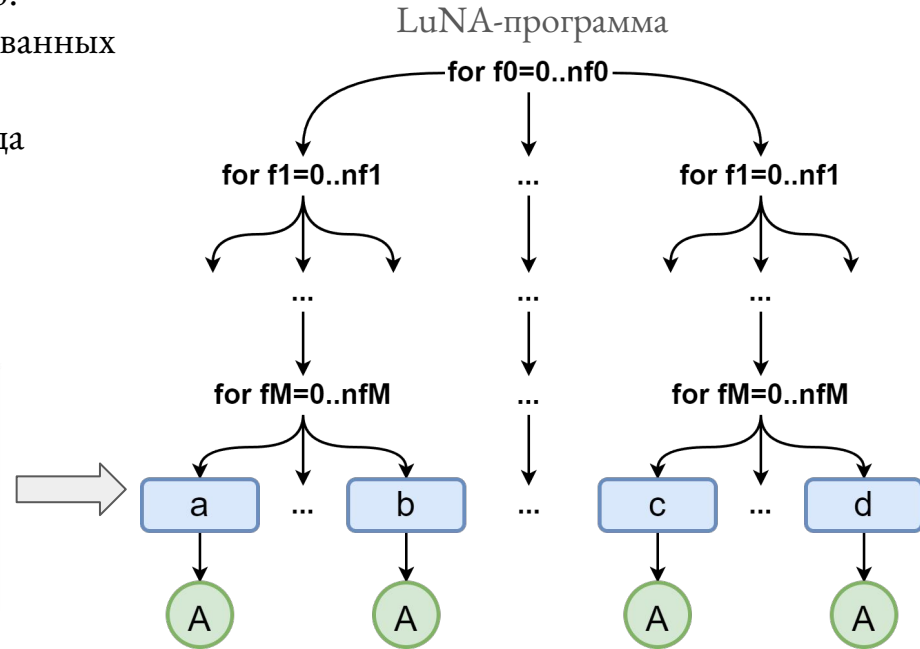
Для выполнения фрагментации по пространству нужно:

- Определить фрагменты кода исходя из преобразованных циклов исходной программы.
- Описать фрагменты вычислений с помощью гнезда массовых операторов `for`.

Исходная программа

```

01: for (int i0 = 0; i0 < N0; ++i0)
02:   for (int i1 = 0; i1 < N1; ++i1)
03:     ...
04:     for (int iM = 0; iM < NM; ++iM)
05:       // вычисление элемента массива без побочных
06:       // эффектов
07:       A[i0][i1]...[iM] = calc_no_side_effects(i0, i1, ...,
08:                                               iM);
  
```



Фрагментация по пространству — параллельное выполнение циклов

Массивы тоже могут фрагментироваться по пространству, то есть хранится ни в одном, а в **нескольких** фрагментах данных.

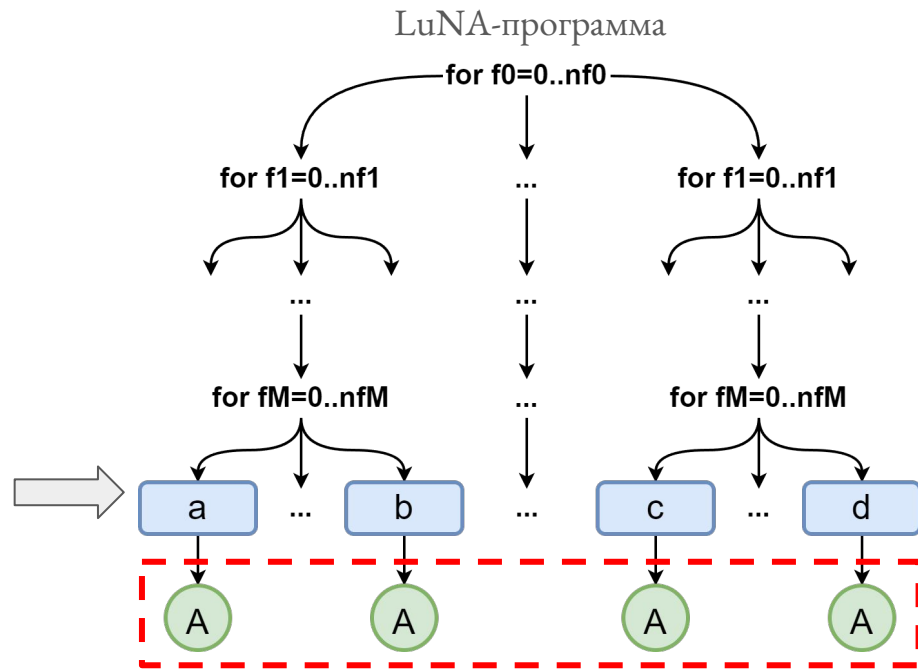
При фрагментации массивов по пространству может возникнуть потребность использования **теневых** граней.

Исходная программа

```

01: for (int i0 = 0; i0 < N0; ++i0)
02:   for (int i1 = 0; i1 < N1; ++i1)
03:     ...
04:     for (int iM = 0; iM < NM; ++iM)
05:       // вычисление элемента массива без побочных
06:       // эффектов
07:       A[i0][i1]...[iM] = calc_no_side_effects(i0, i1, ...,
08:                                               iM);

```



Метод Якоби. Исходная программа

Генератор обладает информацией о сигнатурах определенных атомарных фрагментов кода

```

01:  #define L 384
02:  #define ITMAX 100
03:  ...
04:  double B[L][L][L];
05:  double A[L][L][L];
06:
07:  int main(int an, char **as)
08:  {
09:      int i, j, k, it;
10:  1 double eps, MAXEPS = 0.5f;
11:      for (i = 0; i < L; i++)
12:          for (j = 0; j < L; j++)
13:              for (k = 0; k < L; k++)
14:                  {
15:                      A[i][j][k] = 0;
16:                      if (i == 0 || j == 0
17:  2 || k == 0 || i == L - 1
18:                      || j == L - 1 || k == L - 1)
19:                          B[i][j][k] = 0;
20:                      else
21:                          B[i][j][k] = 4 + i + j + k;
22:                  }
23:      for (it = 1; it <= ITMAX; it++)
24:          {
25:  3 eps = 0;

```

```

26:      for (i = 1; i < L - 1; i++)
27:          for (j = 1; j < L - 1; j++)
28:              for (k = 1; k < L - 1; k++)
29:                  {
30:  4 double tmp = fabs(B[i][j][k] - A[i][j][k]);
31:                      eps = Max(tmp, eps);
32:                      A[i][j][k] = B[i][j][k];
33:                  }
34:
35:      for (i = 1; i < L - 1; i++)
36:          for (j = 1; j < L - 1; j++)
37:              for (k = 1; k < L - 1; k++)
38:  5 B[i][j][k] = (A[i - 1][j][k] + A[i][j - 1][k]
39:                  + A[i][j][k - 1] + A[i][j][k + 1]
40:                  + A[i][j + 1][k] + A[i + 1][j][k]) / 6.0;
41:
42:  6 printf(" IT = %4i  EPS = %14.7E\n", it, eps);
43:      if (eps < MAXEPS)
44:          break;
45:      }
46:  7 ... // вывод результатов работы программы
47:      return 0;
48:  }
49:
50:

```

* 2, 4 и 5 определяются с учетом фрагментации по пространству

Метод Якоби. LuNA-программа

Фрагмент кода main и объявлений фрагментов данных

```
23: ...
24:
25: sub main(int nf, int np)
26: {
27:     df A_unmerged, A, B, MAXEPS, fragmented_eps, eps, print_next, final_it,
28:         A_shadow_1_0_0, A_shadow_m1_0_0, A_shadow_0_1_0, A_shadow_0_m1_0, A_shadow_0_0_1, A_shadow_0_0_m1,
29:         A_shadow_1_1_0, A_shadow_m1_1_0, A_shadow_1_m1_0, A_shadow_m1_m1_0,
30:         A_shadow_0_1_1, A_shadow_0_m1_1, A_shadow_0_1_m1, A_shadow_0_m1_m1,
31:         A_shadow_1_0_1, A_shadow_m1_0_1, A_shadow_1_0_m1, A_shadow_m1_0_m1,
32:         A_shadow_1_1_1, A_shadow_1_1_m1, A_shadow_1_m1_1, A_shadow_1_m1_m1,
33:         A_shadow_m1_1_1, A_shadow_m1_1_m1, A_shadow_m1_m1_1, A_shadow_m1_m1_m1;
34:
35: ...
```

У main есть два параметра — количество фрагментов вычислений, в рамках которых будут выполняться фрагментированные по пространству циклы, и количество узлов.

Метод Якоби. LuNA-программа

Инициализация массивов A и B

```

08:  ...
09:
10:  double eps, MAXEPS = 0.5f;
11:  for (i = 0; i < L; i++)
12:    for (j = 0; j < L; j++)
13:      for (k = 0; k < L; k++)
14:        {
15:          A[i][j][k] = 0;
16:          if (i == 0 || j == 0
17:              || k == 0 || i == L - 1
18:              || j == L - 1 || k == L - 1)
19:            B[i][j][k] = 0;
20:          else
21:            B[i][j][k] = 4 + i + j + k;
22:        }
23:
24:  ...
25:

```

```

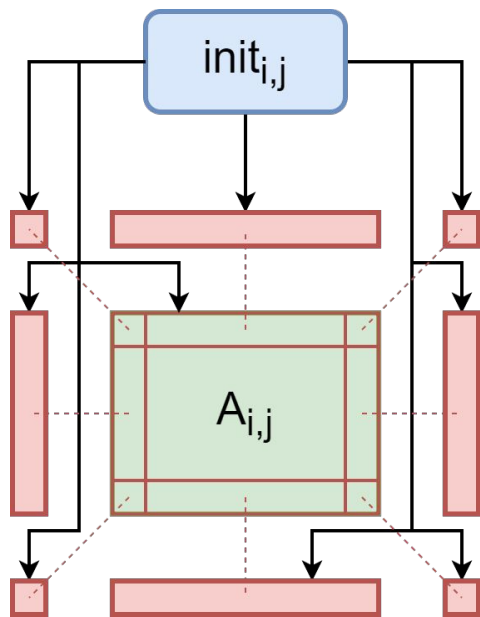
33:  ...
34:
35:  ( code_1(MAXEPS) );
36:
37:  for f0=0..nf-1
38:    for f1=0..nf-1
39:      for f2=0..nf-1
40:        {
41:          loop_2(f0, nf, f1, nf, f2, nf,
42:                A_unmerged[0][0][f0][f1][f2], B[0][0][f0][f1][f2],
43:                A_shadow_0_0_1[0][0][f0][f1][f2],
44:                A_shadow_0_0_m1[0][0][f0][f1][f2],
45:                A_shadow_0_1_0[0][0][f0][f1][f2],
54:                ...) ...;
58:
59:  ...
60:
61:

```

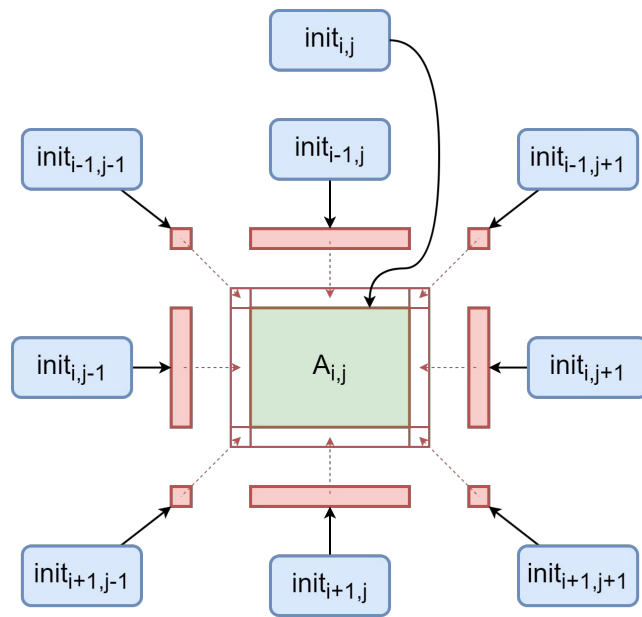
Теневые грани

Инициализация и объединение теневых граней массива A .

Инициализация



Объединение



Синие прямоугольники — фрагменты вычислений, зеленые — массив A , красные — теневые грани

Метод Якоби. LuNA-программа

Слияние теневых граней

```

08:  ...
09:
10:  double eps, MAXEPS = 0.5f;
11:  for (i = 0; i < L; i++)
12:    for (j = 0; j < L; j++)
13:      for (k = 0; k < L; k++)
14:        {
15:          A[i][j][k] = 0;
16:          if (i == 0 || j == 0
17:              || k == 0 || i == L - 1
18:              || j == L - 1 || k == L - 1)
19:            B[i][j][k] = 0;
20:          else
21:            B[i][j][k] = 4 + i + j + k;
22:        }
23:
24:  ...
25:

```

В дальнейшем для вычисления массива В потребуются теньевые грани.

```

33:  ...
34:
35:  code_1(MAXEPS);
36:
37:  for f0=0..nf-1
38:    for f1=0..nf-1
39:      for f2=0..nf-1
40:        {
41:          loop_2(f0, nf, f1, nf, f2, nf,
42:                A_unmerged[0][0][f0][f1][f2], B[0][0][f0][f1][f2],
43:                A_shadow_0_0_1[0][0][f0][f1][f2],
44:                A_shadow_0_0_m1[0][0][f0][f1][f2],
45:                A_shadow_0_1_0[0][0][f0][f1][f2],
46:                ...) ...;
47:
48:          merge_shadow_loop_2_A(f0, nf, f1, nf, f2, nf,
49:                                A_shadow_0_0_m1[0][0][f0][f1][(nf + f2 - 1) % nf],
50:                                A_shadow_0_0_1[0][0][f0][f1][(f2 + 1) % nf],
51:                                A_shadow_0_m1_0[0][0][f0][(nf + f1 - 1) % nf][f2],
52:                                ...,
53:                                A_unmerged[0][0][f0][f1][f2], A[0][0][f0][f1][f2])
54:        }
55:
56:  ...
57:
58:  ...
59:
60:  ...
61:
62:  ...
63:
64:  ...
65:
66:  ...
67:
68:  ...
69:
70:  ...
71:
72:  ...
73:
74:  ...
75:
76:  ...
77:

```

Метод Якоби. LuNA-программа

Рекомендации

```

08:  ...
09:
10:  double eps, MAXEPS = 0.5f;
11:  for (i = 0; i < L; i++)
12:    for (j = 0; j < L; j++)
13:      for (k = 0; k < L; k++)
14:        {
15:          A[i][j][k] = 0;
16:          if (i == 0 || j == 0
17:              || k == 0 || i == L - 1
18:              || j == L - 1 || k == L - 1)
19:            B[i][j][k] = 0;
20:          else
21:            B[i][j][k] = 4 + i + j + k;
22:        }
23:
24:  ...
25:

```

```

#define LOC3(f0, f1, f2, nf0, nf1, nf2, np) (((f0)*($nf1)*($nf2) +
($f1)*($nf2) + ($f2))*($np) / (($nf0)*($nf1)*($nf2))

34:  ...
35:  code_1(MAXEPS);
36:
37:  for f0=0..nf-1
38:    for f1=0..nf-1
39:      for f2=0..nf-1
40:        {
41:          ...
58:          merge_shadow_loop_2_A(f0, nf, f1, nf, f2, nf,
59:            A_shadow_0_0_m1[0][0][f0][f1][(nf + f2 - 1) % nf],
60:            A_shadow_0_0_1[0][0][f0][f1][(f2 + 1) % nf],
61:            A_shadow_0_m1_0[0][0][f0][(nf + f1 - 1) % nf][f2],
62:            ...,
63:            A_unmerged[0][0][f0][f1][f2], A[0][0][f0][f1][f2]) @ {
64:            delete A_shadow_0_0_m1[0][0][f0][f1][(nf + f2 - 1) % nf];
65:            delete A_shadow_0_0_1[0][0][f0][f1][(f2 + 1) % nf];
66:            delete A_shadow_0_m1_0[0][0][f0][(nf + f1 - 1) % nf][f2];
67:            ...
90:            delete A_unmerged[0][0][f0][f1][f2];
91:            locator_cyclic: $LOC3(f0, f1, f2, nf, nf, nf, np);
92:          };
93:        }
94:  ...

```

Метод Якоби. LuNA-программа

Цикл с основной вычислительной нагрузкой

```

21:  ...
22:
23:  for (it = 1; it <= ITMAX; it++)
24:  {
25:      eps = 0;
26:      for (i = 1; i < L - 1; i++)
27:          for (j = 1; j < L - 1; j++)
28:              for (k = 1; k < L - 1; k++)
29:                  {
30:                      double tmp = fabs(B[i][j][k] - A[i][j][k]);
31:                      eps = Max(tmp, eps);
32:                      A[i][j][k] = B[i][j][k];
33:                  }
34:
35:      for (i = 1; i < L - 1; i++)
36:          for (j = 1; j < L - 1; j++)
37:              for (k = 1; k < L - 1; k++)
38:                  B[i][j][k] = (A[i - 1][j][k] + A[i][j - 1][k] +
39:                               A[i][j][k - 1] + A[i][j][k + 1] +
40:                               A[i][j + 1][k] + A[i + 1][j][k]) / 6.0;
41:
42:      printf(" IT = %4i   EPS = %14.7E\n", it, eps);
43:      if (eps < MAXEPS)
44:          break;
45:  }
46:
47:  ...

```

```

115:  while (it <= 100 &&
116:         eps[0][it - 1][nf * nf * nf] >= MAXEPS),
117:         it = 1..out final_it
118:  {
119:      → code_3(eps[0][it][0]);
120:
121:      for f0=0..nf-1
122:          for f1=0..nf-1
123:              for f2=0..nf-1
124:                  {
125:                      loop_4(...) @ ...
126:                      merge_shadow_loop_4_A(...) @ ...
127:                      reduction_max(
128:                          fragmented_eps[0][it][f0][f1][f2],
129:                          eps[0][it][f0 * nf * nf + f1 * nf + f2],
130:                          eps[0][it][f0 * nf * nf + f1 * nf + f2 + 1]);
131:                  }
132:
133:      for f0=0..nf-1
134:          for f1=0..nf-1
135:              for f2=0..nf-1
136:                  {
137:                      loop_5(f0, nf, f1, nf, f2, nf,
138:                             A[1][it][f0][f1][f2],
139:                             B[1][it - 1][f0][f1][f2],
140:                             B[1][it][f0][f1][f2]) @ ...
141:                  }
142:      ...

```

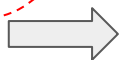
Фрагментация по времени — представление множественного присваивания

Предлагается при именовании фрагментов данных использовать **временные индексы**.

Временные индексы соответствуют порядку выполнения фрагментов вычислений:
 i -й временной индекс фрагмента данных df равен количеству использований фрагмента данных df в качестве выходного фрагмента данных на уровне вложенности i .

Исходная программа

```
01: int main()
02: {
03:   int a;
04:   a = 3; // запись значения переменной a
05:   for (int i = 0; i < 50; ++i)
06:   {
07:     a *= 2; // чтение и запись значения
08:             // переменной a
09:   }
10: }
```



LuNA-программа

```
01: sub main() {
02:   df a;
03:   init_a(a[0]);
04:   destructive_consumption(a[0], a[1][-1]);
05:   for i=0..49 {
06:     loop_body(a[1][i], a[1][i-1]);
07:   }
08:   destructive_consumption(a[1][49], a[2][-1]);
09: }
```

Метод Якоби. LuNA-программа

Временные индексы

```

21:  ...
22:
23:  for (it = 1; it <= ITMAX; it++)
24:  {
25:      eps = 0;
26:      for (i = 1; i < L - 1; i++)
27:          for (j = 1; j < L - 1; j++)
28:              for (k = 1; k < L - 1; k++)
29:                  {
30:                      double tmp = fabs(B[i][j][k] - A[i][j][k]);
31:                      eps = Max(tmp, eps);
32:                      A[i][j][k] = B[i][j][k];
33:                  }
34:
35:      for (i = 1; i < L - 1; i++)
36:          for (j = 1; j < L - 1; j++)
37:              for (k = 1; k < L - 1; k++)
38:                  B[i][j][k] = (A[i - 1][j][k] + A[i][j - 1][k] +
39:                               A[i][j][k - 1] + A[i][j][k + 1] +
40:                               A[i][j + 1][k] + A[i + 1][j][k]) / 6.0;
41:
42:      printf(" IT = %4i  EPS = %14.7E\n", it, eps);
43:      if (eps < MAXEPS)
44:          break;
45:  }
46:
47:  ...

```

```

115:  while (it <= 100 &&
116:         eps[0][it - 1][nf * nf * nf] >= MAXEPS),
117:         it = 1..out final_it
118:  {
119:      → code_3(eps[0][it][0]);
120:
121:      for f0=0..nf-1
122:          for f1=0..nf-1
123:              for f2=0..nf-1
124:                  {
125:                      loop_4(...) @ ...
126:                      merge_shadow_loop_4_A(...) @ ...
127:                      reduction_max(
128:                          fragmented_eps[0][it][f0][f1][f2],
129:                          eps[0][it][f0 * nf * nf + f1 * nf + f2],
130:                          eps[0][it][f0 * nf * nf + f1 * nf + f2 + 1]);
131:                  }
132:
133:      for f0=0..nf-1
134:          for f1=0..nf-1
135:              for f2=0..nf-1
136:                  {
137:                      loop_5(f0, nf, f1, nf, f2, nf,
138:                          A[1][it][f0][f1][f2],
139:                          B[1][it - 1][f0][f1][f2],
140:                          B[1][it][f0][f1][f2]) @ ...
141:                  }
142:      ...

```

Метод Якоби. LuNA-программа

Редукционные операции

```

21:  ...
22:
23:  for (it = 1; it <= ITMAX; it++)
24:  {
25:      eps = 0;
26:      for (i = 1; i < L - 1; i++)
27:          for (j = 1; j < L - 1; j++)
28:              for (k = 1; k < L - 1; k++)
29:                  {
30:                      double tmp = fabs(B[i][j][k] - A[i][j][k]);
31:                      eps = Max(tmp, eps);
32:                      A[i][j][k] = B[i][j][k];
33:                  }
34:
35:      for (i = 1; i < L - 1; i++)
36:          for (j = 1; j < L - 1; j++)
37:              for (k = 1; k < L - 1; k++)
38:                  B[i][j][k] = (A[i - 1][j][k] + A[i][j - 1][k] +
39:                               A[i][j][k - 1] + A[i][j][k + 1] +
40:                               A[i][j + 1][k] + A[i + 1][j][k]) / 6.0;
41:
42:      printf(" IT = %4i  EPS = %14.7E\n", it, eps);
43:      if (eps < MAXEPS)
44:          break;
45:  }
46:
47:  ...

```

```

115:  while (it <= 100 &&
116:         eps[0][it - 1][nf * nf * nf] >= MAXEPS),
117:         it = 1..out final_it
118:  {
119:      code_3(eps[0][it][0]);
120:
121:      for f0=0..nf-1
122:          for f1=0..nf-1
123:              for f2=0..nf-1
124:                  {
125:                      loop_4(...) @ ... // выполнение фрагмента гнезда
126:                      merge_shadow_loop_4_A(...) @ ... // слияние т.г.
127:                      reduction_max(
128:                          fragmented_eps[0][it][f0][f1][f2],
129:                          eps[0][it][f0 * nf * nf + f1 * nf + f2],
130:                          eps[0][it][f0 * nf * nf + f1 * nf + f2 + 1]);
131:                  }
132:
133:      for f0=0..nf-1
134:          for f1=0..nf-1
135:              for f2=0..nf-1
136:                  {
137:                      loop_5(f0, nf, f1, nf, f2, nf,
138:                             A[1][it][f0][f1][f2],
139:                             B[1][it - 1][f0][f1][f2],
140:                             B[1][it][f0][f1][f2]) @ ...
141:                  }
142:      ...

```

Метод Якоби. LuNA-программа Использование теневых граней

```

21:  ...
22:
23:  for (it = 1; it <= ITMAX; it++)
24:  {
25:      eps = 0;
26:      for (i = 1; i < L - 1; i++)
27:          for (j = 1; j < L - 1; j++)
28:              for (k = 1; k < L - 1; k++)
29:              {
30:                  double tmp = fabs(B[i][j][k] - A[i][j][k]);
31:                  eps = Max(tmp, eps);
32:                  A[i][j][k] = B[i][j][k];
33:              }
34:
35:      for (i = 1; i < L - 1; i++)
36:          for (j = 1; j < L - 1; j++)
37:              for (k = 1; k < L - 1; k++)
38:                  B[i][j][k] = (A[i - 1][j][k] + A[i][j - 1][k] +
39:                               A[i][j][k - 1] + A[i][j][k + 1] +
40:                               A[i][j + 1][k] + A[i + 1][j][k]) / 6.0;
41:
42:      printf(" IT = %4i   EPS = %14.7E\n", it, eps);
43:      if (eps < MAXEPS)
44:          break;
45:  }
46:
47:  ...

```

```

115:  while (it <= 100 &&
116:         eps[0][it - 1][nf * nf * nf] >= MAXEPS),
117:         it = 1..out final_it
118:  {
119:      code_3(eps[0][it][0]);
120:
121:      for f0=0..nf-1
122:          for f1=0..nf-1
123:              for f2=0..nf-1
124:              {
125:                  loop_4(...) @ ...
126:                  merge_shadow_loop_4_A(...) @ ...
127:                  reduction_max(
128:                      fragmented_eps[0][it][f0][f1][f2],
129:                      eps[0][it][f0 * nf * nf + f1 * nf + f2],
130:                      eps[0][it][f0 * nf * nf + f1 * nf + f2 + 1]);
131:              }
132:
133:      for f0=0..nf-1
134:          for f1=0..nf-1
135:              for f2=0..nf-1
136:              {
137:                  loop_5(f0, nf, f1, nf, f2, nf,
138:                      A[1][it][f0][f1][f2],
139:                      B[1][it - 1][f0][f1][f2],
140:                      B[1][it][f0][f1][f2]) @ ...
141:              }
142:  ...

```

Метод Якоби. LuNA-программа

Выражения, влияющие на граф потока управления

```

21:  ...
22:
23:  for (it = 1; it <= ITMAX; it++)
24:  {
25:      eps = 0;
26:      for (i = 1; i < L - 1; i++)
27:          for (j = 1; j < L - 1; j++)
28:              for (k = 1; k < L - 1; k++)
29:              {
30:                  double tmp = fabs(B[i][j][k] - A[i][j][k]);
31:                  eps = Max(tmp, eps);
32:                  A[i][j][k] = B[i][j][k];
33:              }
34:
35:      for (i = 1; i < L - 1; i++)
36:          for (j = 1; j < L - 1; j++)
37:              for (k = 1; k < L - 1; k++)
38:                  B[i][j][k] = (A[i - 1][j][k] + A[i][j - 1][k] +
39:                               A[i][j][k - 1] + A[i][j][k + 1] +
40:                               A[i][j + 1][k] + A[i + 1][j][k]) / 6.0;
41:
42:      printf(" IT = %4i  EPS = %14.7E\n", it, eps);
43:      if (eps < MAXEPS)
44:          break;
45:  }
46:
47:  ...

```

```

115:  while (it <= 100 &&
116:         eps[0][it - 1][nf * nf * nf] >= MAXEPS),
117:      it = 1..out final_it
118:  {
119:      code_3(eps[0][it][0]);
120:
121:      for f0=0..nf-1
122:          for f1=0..nf-1
123:              for f2=0..nf-1
124:              {
125:                  loop_4(...) @ ...
126:                  merge_shadow_loop_4_A(...) @ ...
127:                  reduction_max(
128:                      fragmented_eps[0][it][f0][f1][f2],
129:                      eps[0][it][f0 * nf * nf + f1 * nf + f2],
130:                      eps[0][it][f0 * nf * nf + f1 * nf + f2 + 1]);
131:              }
132:
133:      for f0=0..nf-1
134:          for f1=0..nf-1
135:              for f2=0..nf-1
136:              {
137:                  loop_5(f0, nf, f1, nf, f2, nf,
138:                      A[1][it][f0][f1][f2],
139:                      B[1][it - 1][f0][f1][f2],
140:                      B[1][it][f0][f1][f2]) @ ...
141:              }
142:  ...

```

Метод Якоби. LuNA-программа

Упорядочение фрагментов вычислений

```

21:  ...
22:
23:  for (it = 1; it <= ITMAX; it++)
24:  {
25:      eps = 0;
26:      for (i = 1; i < L - 1; i++)
27:          for (j = 1; j < L - 1; j++)
28:              for (k = 1; k < L - 1; k++)
29:                  {
30:                      double tmp = fabs(B[i][j][k] - A[i][j][k]);
31:                      eps = Max(tmp, eps);
32:                      A[i][j][k] = B[i][j][k];
33:                  }
34:
35:      for (i = 1; i < L - 1; i++)
36:          for (j = 1; j < L - 1; j++)
37:              for (k = 1; k < L - 1; k++)
38:                  B[i][j][k] = (A[i - 1][j][k] + A[i][j - 1][k] +
39:                               A[i][j][k - 1] + A[i][j][k + 1] +
40:                               A[i][j + 1][k] + A[i + 1][j][k]) / 6.0;
41:
42:      printf(" IT = %4i  EPS = %14.7E\n", it, eps);
43:      if (eps < MAXEPS)
44:          break;
45:  }
46:
47:  ...

```

```

113:  init_int(1, print_next[0][1]);
114:
115:  while (it <= 100 &&
116:         eps[0][it - 1][nf * nf * nf] >= MAXEPS),
117:         it = 1..out final_it
118:  {
119:      ...
120:
121:      if (print_next[0][it])
122:      {
123:          code_6(eps[0][it][nf * nf * nf], it) >>
124:              (print_next[0][it + 1]);
125:      }
126:  }
127:
128:  ...

```

В данном случае вывод должен выполняться упорядоченно, но это не следует из информационных зависимостей.

Заключение

Алгоритмы генератора фрагментов вычислений и данных были разработаны на языке Python.

В дальнейшем планируется:

- Реализовать алгоритмы определения информации, необходимой для генерации LuNA-программ.
- Реализовать алгоритмы анализа последовательных программ.
- Развить разработанные алгоритмы и представления таким образом, чтобы они подходили для более широкого класса задач.