М.А. Горелов (МФТИ (ГУ))

M. Gorelov

ИСПОЛЬЗОВАНИЕ АФФИННЫХ ПРЕОБРАЗОВАНИЙ ДЛЯ АВТОМАТИЧЕСКОГО РАСПАРАЛЛЕЛИВАНИЯ ЦИКЛОВ

USING OF AFFINE TRANSFORMATIONS FOR AUTOMATIC LOOP PARALLELIZATION

Рассмотрены методы повышения эффективности автоматического распараллеливания в оптимизирующем компиляторе «Эльбрус». В их основе лежит трансформация циклов с зависимостями между итерациями к виду, в котором зависимости отсутствуют. Представлен анализ, позволяющий получить необходимые для трансформации аффинные преобразования. Теория основана на методах линейной алгебры и целочисленного линейного программирования.

Ключевые слова: оптимизирующий компилятор, автоматическое распараллеливание, аффинные преобразования.

The article describes methods to improve the efficiency of automatic parallelization in Elbrus optimizing compiler. They are based on transformations of loops with dependences between iterations to the form in which there are no dependences. Presented analysis allows to find necessary affine transformations. The theory is based on the methods of linear algebra and integer linear programming.

Keywords: optimizing compiler, automatic parallelization, affine transformations.

Введение

В настоящее время, когда лидирующие производители микропроцессоров перешли на создание многоядерных архитектур, притом что значительная часть существующих приложений написаны для последовательного исполнения на одноядерных машинах, всё

большую актуальность приобретает проблема автоматического распараллеливания на уровне потоков. Она особенно значима в силу того, что в большинстве вычислительных задач основная часть времени тратится на вычисления внутри циклов, где оптимизация этой категории может дать существенный эффект. В то же время существующая в компиляторе «Эльбрус» техника автоматического распараллеливания не решает проблему распараллеливания циклов с зависимостями между итерациями, что и послужило причиной для данной работы.

Основное внимание в статье сосредоточено на методах оптимизации класса численных приложений, в которых обращения, выполняемые по отношению к индексам охватывающих циклов, представимы в виде аффинных функций. Функция от одной или нескольких переменных называется аффинной, если она может быть представлена как линейная комбинация этих переменных плюс константа. Например, если i и j – индексные переменные охватывающих циклов, то $X[i][j]$ и $X[i][2 \cdot i+j]$ – аффинные обращения.

Простой пример цикла с аффинными обращениями к данным имеет вид:

```
for (i = 0; i < 10; i++)
 A[i] = 0;
```

Поскольку в итерациях цикла производится запись в разные ячейки памяти, разные итерации могут одновременно выполняться разными процессорами. Однако, если бы при этом выполнялась другая инструкция, скажем, $A[j] = 1$, то надо было бы отслеживать, не может ли индекс i быть равным j, и, если может, выяснять, в каком порядке следует выполнять экземпляры этих двух инструкций.

Крайне важно знать, какие итерации могут обращаться к одним и тем же ячейкам памяти. Это позволяет определить зависимости данных, которые следует учитывать при планировании кода как для однопроцессорных, так и для многопроцессорных систем. Цель алгоритмов, представленных в работе, заключается в том, чтобы найти план, согласно которому операции, обращающиеся к одной и той же ячейке памяти, выполнялись бы по возможности ближе друг к другу, причем в многопроцессорной системе – на одном и

том же процессоре.

1. Элементы теории аффинных преобразований

Основные понятия

Согласно [1], в задачах оптимизации циклов с обращениями к массивам используются три вида пространств, каждое из которых можно рассматривать как точки одномерной или многомерной структуры.

- *пространство итераций* представляет собой множество комбинаций значений, которые принимают индексы циклов;
- *пространство данных* представляет собой множество элементов массива, к которым выполняется обращение;
- *пространство процессоров* представляет собой множество процессоров в системе; обычно этим процессорам назначены целочисленные номера или векторы, для того чтобы отличать один от другого.

В качестве входных данных оптимизации подаются: последовательный порядок, в котором выполняются итерации, и аффинные функции обращения к массивам, указывающие порядок обращения экземпляров из пространства итераций к элементам из пространства данных. Выходные данные оптимизации представляют собой аффинные функции, определяющие, что и когда должен делать каждый процессор.

Пример:

```
float Z[100];
for (i = 0; i < 10; i++)
  Z[i+10] = Z[i];
```

Для данного примера три пространства выглядят следующим образом (рис. 1):

- пространство итераций: одномерное пространство из десяти итераций, помеченных значениями индекса $i = 0, 1,\ldots, 9$;
- пространство данных задается непосредственно объявлениями массивов. Здесь элементы массива проиндексированы значениями $a = 0, 1,...,99$;

• пространство процессоров. Пусть в целевой системе имеется неограниченное количество виртуальных процессоров. Если после распараллеливания оказывается, что физических процессоров меньше, чем виртуальных, то виртуальные процессоры разделяются на одинаковые блоки, и каждый блок назначается физическому процессору. В данном примере необходимо десять виртуальных процессоров, по одному для каждой итерации цикла.

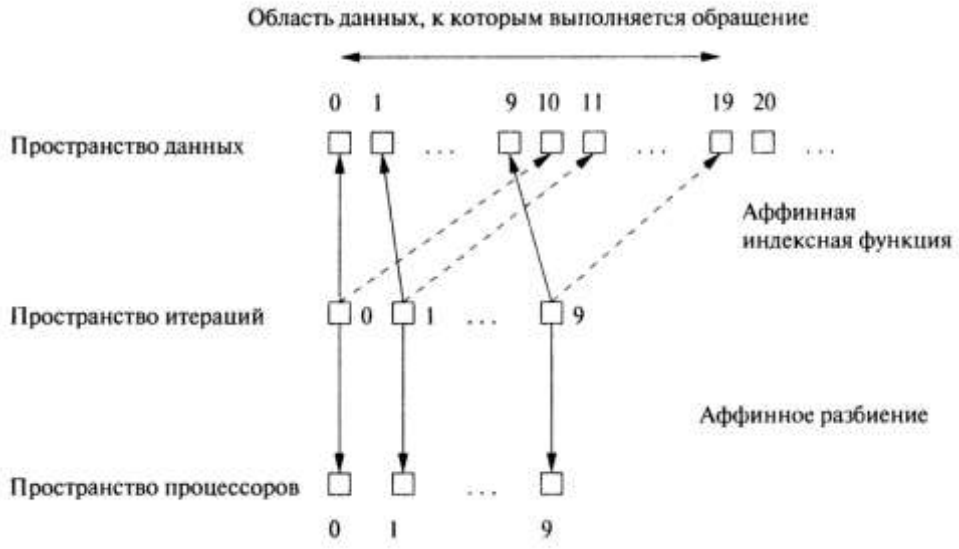

Рис. 1

Пространства итераций, данных и процессоров

Справедливы следующие отображения между рассматриваемыми пространствами:

• *аффинная индексная функция*. Каждое обращение к массиву определяет отображение итерации из пространства итераций на элемент массива в пространстве данных;

• *аффинное разбиение*. Цикл распараллеливается с использованием аффинной функции для назначения итераций (из пространства итераций) процессорам (из пространства процессоров). В нашем примере итерация i назначается процессору i;

• *область данных, к которым выполняется обращение*, получается путем объединения информации о пространстве итераций с индексной функцией массива. В примере обращение к массиву $Z[i+10]$ затрагивает область $\{a \mid 10 \le a < 20\}$, а обращение $Z[i]$ – область $\{a \mid 0 \le a < 10\}$;

• *зависимости данных*. Для определения того, можно ли распараллелить данный

цикл, следует выяснить, пересекают ли зависимости данных границы каждой итерации. Во взятом примере все итерации независимы и могут быть выполнены параллельно, в любом выбранном порядке.

Формулировка задачи в терминах многомерных пространств и аффинных отображений между ними позволяет применить стандартный математический аппарат для решения задачи в общем виде.

Матричная запись

Математически итерации цикла глубиной d могут быть представлены как

$$\{\boldsymbol{i} \in Z^d \mid \boldsymbol{B}\boldsymbol{i} + \boldsymbol{b} \geq \boldsymbol{0}\},$$

где Z – множество целых чисел; $\boldsymbol{B}$ – целочисленная матрица размером $2d \times d$; $\boldsymbol{b}$ – целочисленный вектор длиной $2d$; $\boldsymbol{0}$ – нулевой вектор длиной $2d$.

Тогда пространство итераций из предыдущего примера можно записать в виде

$$\begin{bmatrix} 1 \\ -1 \end{bmatrix} i + \begin{bmatrix} 0 \\ 9 \end{bmatrix} \geq 0.$$

Аффинные функции обращения к данным также могут быть представлены в матричном виде. Каждое аффинное обращение к массиву можно описать двумя матрицами и двумя векторами. Первая пара «матрица – вектор», $\boldsymbol{B}$ и $\boldsymbol{b}$, описывает пространство итераций для данных обращений. Вторая пара, $\boldsymbol{F}$ и $\boldsymbol{f}$, представляет функции от индексных переменных циклов, которые возвращают индексы массива, используемые в качестве различных измерений при обращении к массиву.

Формально мы представляем обращения к массиву в гнезде циклов с вектором индексных переменных $\boldsymbol{i}$ в виде четверки $F = <\boldsymbol{F}, \boldsymbol{f}, \boldsymbol{B}, \boldsymbol{b}>$. Она отображает вектор $\boldsymbol{i}$ в границах $\boldsymbol{B}\boldsymbol{i} + \boldsymbol{b} \geq \boldsymbol{0}$ на элемент массива $\boldsymbol{F}\boldsymbol{i} + \boldsymbol{f}$.

2. Оптимизация реальных программ: поиск параллельности, не требующей синхронизации

В первую очередь приведенные элементы теории можно использовать применительно к задаче распараллеливания для случая, когда между процессорами нет никакого взаимодействия или синхронизации. Хотя это ограничение может представиться как чисто академический пример (такие программы встречаются в реальности), и алгоритм решения поставленной задачи оказывается весьма полезным.

Разбиения аффинного пространства

Чтобы устранить взаимодействие, каждая пара операций с зависимостями данных должна быть назначена одному и тому же процессору. С этой целью для каждой инструкции следует найти разбиение аффинного пространства, т.е. разделить пространство итераций на независимые подпространства, каждое из которых выполняется одним процессором. Цель состоит в создании максимально возможного количества независимых разделов, удовлетворяющих ограничениям разбиения пространства.

Формально аффинное разбиение программы не требует синхронизации тогда и только тогда, когда для каждых двух (не обязательно различных) зависимых обращений $F_1 = <\boldsymbol{F_1}, \boldsymbol{f_1}, \boldsymbol{B_1}, \boldsymbol{b_1}>$ в инструкции s_1, вложенной в d_1 циклов, и $F_2 = <\boldsymbol{F_2}, \boldsymbol{f_2}, \boldsymbol{B_2}, \boldsymbol{b_2}>$ в инструкции s_2, вложенной в d_2 циклов, разбиения $<\boldsymbol{C_1}, \boldsymbol{c_1}>$ и $<\boldsymbol{C_2}, \boldsymbol{c_2}>$ инструкций s_1 и s_2, соответственно, удовлетворяют следующим ограничениям пространства: для всех $\boldsymbol{i_1}$ из Z^{d}_1 и $\boldsymbol{i_2}$ из Z^{d}_2, таких, что

$$\boldsymbol{B_1 i_1} + \boldsymbol{b_1} \geq 0, \quad \boldsymbol{B_2 i_2} + \boldsymbol{b_2} \geq 0, \quad \boldsymbol{F_1 i_1} + \boldsymbol{f_1} = \boldsymbol{F_2 i_2} + \boldsymbol{f_2},$$

выполняется соотношение $\boldsymbol{C_1 i_1} + \boldsymbol{c_1} = \boldsymbol{C_2 i_2} + \boldsymbol{c_2}$.

Пример. Рассмотрим гнездо:

```
for (i = 1; i <= 100; i++)
 for (j = 1; j <= 100; j++) {
       X[i,j] = X[i,j] + Y[i-1,j]; // s1
       Y[I,j] = Y[i,j] + X[i,j-1]; // s2
        }
```

Из диаграммы на рис. 2 видно, что данный код можно распараллелить без синхронизации, назначив каждую цепочку зависимых операций одному и тому же процессору. Поскольку имеется две инструкции, необходимо найти два аффинных разбиения, по одному

для каждой инструкции. Каждое из них можно представить матрицей 1×2 и вектором 1×1, которые транслируют вектор индексов [***i, j***] в единственный номер процессора. Пусть <[$\boldsymbol{C_{11}C_{12}}$], [$\boldsymbol{c_1}$]> и <[$\boldsymbol{C_{21}C_{22}}$], [$\boldsymbol{c_2}$]> – это одномерные аффинные разбиения для инструкций $\boldsymbol{s}_1$ и $\boldsymbol{s}_2$ соответственно.

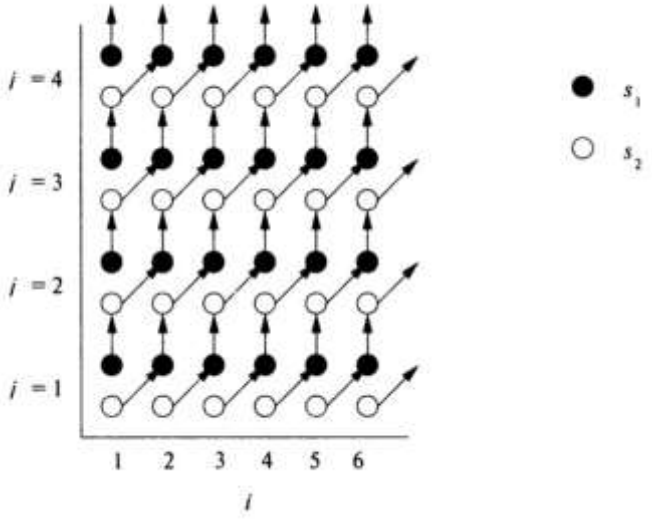

Рис. 2

Зависимости в рассматриваемом коде

В рассматриваемом коде присутствуют две зависимости – между записью $X[i,j]$ в инструкции s_1 и чтением $X[i,j\text{-}1]$ в инструкции s_2 и между записью $Y[i,j]$ в инструкции s_2 и чтением $Y[i\text{-}1,j]$ в инструкции s_1. Ограничения разбиений пространства, обусловленные зависимостями данных между $X[i,j]$ в инструкции s_1 и $X[i,j\text{-}1]$ в инструкции s_2, можно выразить следующим образом: для всех (i, j) и (i', j'), таких, что

$$1 \le i \le 100, \quad 1 \le j \le 100, \quad 1 \le i' \le 100, \quad 1 \le j' \le 100, \quad i = i', \quad j = j' - 1,$$

справедливо $$\begin{bmatrix} C_{11} & C_{12} \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} + \begin{bmatrix} c_1 \end{bmatrix} = \begin{bmatrix} C_{21} & C_{22} \end{bmatrix} \begin{bmatrix} i' \\ j' \end{bmatrix} + \begin{bmatrix} c_2 \end{bmatrix}.$$

Аналогично можно вывести ограничения разбиений пространства для второй зависимости.

Если уменьшить количество неизвестных, воспользовавшись уравнениями из функций обращений: $t_1 = i = i'$ и $t_2 = j = j - 1$, то уравнение для разбиения превращается в

$$\begin{bmatrix} C_{11} - C_{12} & C_{12} - C_{22} \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} + \begin{bmatrix} c_1 - c_2 - C_{22} \end{bmatrix} = 0 .$$

Ввиду того что это уравнение выполняется при любых t_1 и t_2, справедливы соотношения:

$$\begin{aligned} C_{11} - C_{21} &= 0 \\ C_{12} - C_{22} &= 0 \\ c_1 - c_2 - C_{22} &= 0 \end{aligned} .$$

Для ограничений, связанных с обращениями к Y, получим

$$\begin{aligned} C_{11} - C_{21} &= 0 \\ C_{12} - C_{22} &= 0 \\ c_1 - c_2 + C_{21} &= 0 \end{aligned} .$$

Отсюда следует $C_{11} = C_{21} = -C_{22} = -C_{12} = c_2 - c_1$.

Найдем одно из решений. Если для простоты принять $C_{11} = 1$, то получим: $C_{21} = 1$, $C_{22} = -1$ и $C_{12} = -1$. В качестве постоянных членов для простоты принимаем $c_2 = 0$, тогда $c_1 = -1$.

Пусть p – идентификатор процессора, выполняющего итерацию (i, j). При использовании этого обозначения аффинное разбиение принимает вид:

$$\begin{aligned} s_1 : [p] &= \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} + [-1] \\ s_2 : [p] &= \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} + [0] \end{aligned} .$$

Иначе говоря, (i, j)-я итерация s_1 назначается процессору $p = i - j - 1$, а (i, j)-я итерация s_2 – процессору $p = i - j$. После этого гнездо может быть преобразовано к виду, в котором оно легко распараллеливается:

```
for (p = -100; p <= 99; p++)
 for (i = 1; i <=100; i++)
      for (j = 1; j <= 100; j++) {
           if (p == i - j - 1)
                X[i,j] = X[i,j] + Y[i-1,j]; // s1
           if (p == i - j)
                Y[i.j] = Y[i,j] + X[i,j-1]; // s2
      }
```

3. Оптимизация реальных программ: поиск параллельности с синхронизаций

Постановка задачи

Согласно [2], задача может быть сформулирована следующим образом:

• дано множество операций Ω, где операция – экземпляр инструкции исходного языка программирования на конкретной итерации цикла;

• порядок выполнения операций непроизволен. Задано бинарное отношение Г на множестве Ω (граф зависимостей), такое что:

– $\forall u, v \in \Omega$: $u \Gamma v$, выполнение операции v не может начаться раньше, чем закончится выполнение u;

– обратно, если пара операций (u, v) не удовлетворяет транзитивному замыканию Г, то эти операции могут быть выполнены в любом порядке либо параллельно;

• построение параллельной программы сводится к выбору порядка исполнения операций, максимально близкого к тому, который задается транзитивным замыканием Г.

Простой способ описания параллельной программы – использование «расписания». Расписанием называется функция θ: $\Omega \rightarrow R$, такая что справедливо:

$$u,v \in \Omega: u \Gamma v \Rightarrow \theta(v) \geq \theta(u) + 1.$$

Для любой операции u $\theta(u)$ есть время, когда может начаться выполнение u на компьютере с бесконечным количеством процессоров. Пусть $F(t)$ – множество операций, которые начинают выполняться в момент времени t: $F(t) = \{u \in \Omega \mid \theta(u) = t\}$,

тогда параллельная версия программы будет выглядеть так:

```
do t = 0, L
 doall F(t)
 barrier
end do,
```

где $L = \max_{u \in \Omega} \theta(u)$.

Нахождение расписания

Ввиду того, что операция (u) определяется инструкцией (S) и значением индуктив-

ных переменных (x), справедливо рассматривать расписание для S как функцию от x.

Используя метод неопределенных коэффициентов, будем искать расписание в виде:

$$\theta_S(x) = \underline{\tau_S}\,\vec{x} + \underline{\sigma_S}\,\vec{n} + \alpha\,, \qquad (1)$$

где τ_S, σ_S и α – неизвестные коэффициенты, x – вектор индуктивных переменных, n – вектор символьных констант.

При подстановке этой функции в определение расписания для всех зависимых пар операций образуется система линейных неравенств с неизвестными коэффициентами. В силу определения расписания необходимо добавить неравенства $\theta_S(x) \geq 0$, после чего систему можно решить любым известным способом. Метод, изложенный далее, позволяет упростить ее, используя преимущества описания расписания. Он основан на следующей теореме.

Теорема (аффинная форма леммы Фаркаша). Пусть D – непустой многогранник, заданный системой аффинных неравенств вида: $a_k\vec{x} + b_k \geq 0, k = \overline{1, p}$. В этом случае аффинная функция ψ неотрицательна на D тогда и только тогда, когда она может быть представлена в виде

$$\psi(\vec{x}) \equiv \lambda_0 + \sum_{k=1}^{p} \lambda_k (a_k \vec{x} + b_k), \qquad \lambda_0, ..., \lambda_p \geq 0\,.$$

По теореме аффинная форма вида (1) неотрицательна внутри многогранника пространства итераций тогда и только тогда, когда существуют коэффициенты, такие что

$$\theta_S(x) \equiv \mu_{S_0} + \sum_{k=1}^{m_S} \mu_{S_k} (a_{S_k} \begin{pmatrix} x \\ n \end{pmatrix} + b_{S_k}).$$

Итак, если расписание аффинное, выражение $\theta_{Sy}(y) - \theta_{Sx}(x) - 1$ есть неотрицательная аффинная форма в многограннике объединенного итерационного пространства индексных векторов x, y. Значит, существуют такие коэффициенты λ_{ek}, что

$$\theta_{Sy}(y) - \theta_{Sx}(x) - 1 \equiv \lambda_{e_0} + \sum_{k=1}^{m_e} \lambda_{e_k} (c_{e_k} \begin{pmatrix} x \\ y \\ n \end{pmatrix} + d_{e_k}).$$

Записав уравнения для всех пар зависимостей и избавившись от коэффициентов λ_{ek}, получим систему неравенств для коэффициентов μ_{Sk}. Далее, решив систему, например, с помощью симплекс-метода, получим множество расписаний, удовлетворяющих зависимостям.

Рассмотрим в качестве примера цикл 1-*d* Jacobi [3]:

```
for (t = 0; t < T; t++) {
      for (i = 1; i < N - 1; i++)
       B[i] = 0.33333 * (A[i-1] + A[i] + A[i+1]); //s1
      for (j = 1; j < N - 1; j++)
       A[j] = B[j];                                //s2
    }
```

В данном примере есть две инструкции: s_1 и s_2. Соответственно, для каждой из них можно записать расписания с неопределенными коэффициентами:

$$\theta_1(t,i) = \mu_0 + \mu_1 t + \mu_2(T - t - 1) + \mu_3(i-1) + \mu_4(N - i - 2),$$
$$\theta_2(t,j) = \nu_0 + \nu_1 t + \nu_2(T - t - 1) + \nu_3(j-1) + \nu_4(N - j - 2).$$

Рассмотрим зависимость $B[i'] \rightarrow B[j]$, $i' = j$, $t' = t$.

$$\theta_2(t,j) - \theta_1(t',i') - 1 \geq 0.$$

Подставив выражения для θ_1 и θ_2 и применив теорему, получим уравнение:

$$\begin{aligned}&[\nu_0 + \nu_1 t + \nu_2(T - t - 1) + \nu_3(j-1) + \nu_4(N - j - 2)]\\ &-[\mu_0 + \mu_1 t + \mu_2(T - t - 1) + \mu_3(i-1) + \mu_4(N - i - 2)]\\ &-1 \equiv \lambda_0 + \lambda_1 t + \lambda_2(T - t - 1) + \lambda_3(j-1) + \lambda_4(N - j - 2)\end{aligned}.$$

Приравняв коэффициенты при каждой из переменных, получаем систему уравнений:

$$\begin{cases}\nu_0 + \nu_2 + \nu_3 + 2\nu_4 - \mu_0 + \mu_2 + \mu_3 + 2\mu_4 - 1 = \lambda_0 - \lambda_2 - \lambda_3 - 2\lambda_4\\ \nu_1 - \nu_2 - \mu_1 + \mu_2 = \lambda_1 - \lambda_2\\ \nu_3 - \nu_4 - \mu_3 + \mu_4 = \lambda_3 - \lambda_4\\ \nu_2 - \mu_2 = \lambda_2\\ \nu_4 - \mu_4 = \lambda_4\\ \lambda_i \geq 0, \quad i \in [0,4]\end{cases}.$$

Избавляясь от λ, получим систему неравенств для μ и ν:

$$\begin{cases} \nu_0 - \mu_0 - 1 \geq 0 \\ \nu_1 - \mu_1 \geq 0 \\ \nu_2 - \mu_2 \geq 0 \\ \nu_3 - \mu_3 \geq 0 \\ \nu_4 - \mu_4 \geq 0 \\ \mu_i \geq 0, \quad \nu_j \geq 0, \quad i, j \in [0,4] \end{cases}.$$

Те же действия выполняются для остальных зависимостей. В результате, объединяя полученные данные, имеем для данного цикла следующую систему неравенств:

$$\begin{cases} \nu_1 - \mu_1 = 0 \\ \nu_2 - \mu_2 = 0 \\ \nu_3 - \mu_3 = 0 \\ \nu_4 - \mu_4 = 0 \\ \nu_0 - \mu_0 - 1 \geq 0 \\ \nu_0 - \mu_0 - \mu_3 + \mu_4 - 1 \geq 0 \\ \nu_0 - \mu_0 + \mu_3 - \mu_4 - 1 \geq 0 \\ \mu_1 - \mu_2 - 1 \geq 0 \\ \nu_1 - \nu_2 \geq 0 \\ \mu_0 - \nu_0 + \nu_1 - \nu_2 - 1 \geq 0 \\ \mu_0 - \nu_0 + \nu_1 - \nu_2 + \nu_3 - \nu_4 - 1 \geq 0 \\ \mu_0 - \nu_0 + \nu_1 - \nu_2 - \nu_3 + \nu_4 - 1 \geq 0 \\ \mu_i \geq 0, \quad \nu_j \geq 0, \quad i, j \in [0,4], \end{cases},$$

решая которую любым известным методом, получаем множество решений. Одним из возможных решений является, например,

$$\mu_0 = 0, \quad \mu_1 = 4, \quad \mu_2 = 0, \quad \mu_3 = 1, \quad \mu_4 = 0,$$
$$\nu_0 = 2, \quad \nu_1 = 4, \quad \nu_2 = 0, \quad \nu_3 = 1, \quad \nu_4 = 0.$$

Тогда расписания для инструкций s_1 и s_2 соответственно, принимают вид:

$$\theta_1(t,i) = 4t + i - 1$$
$$\theta_2(t,j) = 4t + j + 1.$$

На основе полученных расписаний исходный цикл можно преобразовать к виду:

```
for (q = 0; q <= 4 * T + 2 * N -5; q++)
 for (t = 0; t < T; t++) {
      if ((q - 4 * t + 1 >= 1) && (q - 4 * t + 1 < N))
                B[q - 4 * t + 1] = 0,33333(A[q - 4 * t]
                  + A[q - 4 * t + 1] + A[q - 4 * t + 2]);     //s1
      if ((q - 4 * t - 1 >= 1) && (q - 4 * t - 1 < N))
                A[q - 4 * t - 1] = B[q - 4 * t - 1];         //s2
```

}

Заметим, что для каждого q все итерации внутреннего цикла по t независимы и могут быть выполнены одновременно на разных процессорах.

Заключение

Модель аффинных преобразований является одной из наиболее перспективных для оптимизаций параллелизма в вычислительных задачах, содержащих большое количество циклов с аффинными обращениями. Методы, рассмотренные выше, достаточно просты и позволяют существенно повысить производительность автоматического распараллеливания. Они разработаны для реализации в оптимизирующем компиляторе Эльбрус на основе уже существующих в нем средств решения систем уравнений.

Литература

1. А. Ахо, М. Лам, Р.Сети, Дж. Ульман. Компиляторы: принципы, технологии и инструментарий, 2-е изд.: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2008, 1184 с.

2. P. Feautrier. Some efficient solutions to the affine scheduling problem. Part I. One-dimensional time. – International Journal of Parallel Programming, 21(5), 1992, p. 313-348.

3. U. Bondhugula. Effective automatic parallelization and locality optimization using the polyhedral model. – 2008.