

Московский физико–технический институт (государственный университет)  
Факультет радиотехники и кибернетики  
Кафедра информатики и вычислительной техники

Выпускная квалификационная работа бакалавра

# Исследование возможности повышения эффективности автоматического распараллеливания в оптимизирующем компиляторе Эльбрус

Выполнил: Горелов Михаил, 913 группа  
Научный руководитель: к.т.н. Муханов Л.Е.

# Актуальность

- ▶ Увеличение производительности вычислительных комплексов, переход на многоядерные архитектуры
- ▶ Значительное количество существующих приложений реализованы для последовательного исполнения
- ▶ В большинстве вычислительных задач основная часть времени тратится на вычисления, которые содержатся внутри циклов либо рекурсивных процедур

# Постановка задачи

- ▶ Исследовать и реализовать алгоритм динамической проверки эффективности применения автоматического распараллеливания
- ▶ Исследовать возможность автоматического распараллеливания рекурсивных процедур и реализовать анализ диапазонов обращения в память
- ▶ Исследовать технику автоматического распараллеливания, базирующуюся на использовании аффинных преобразований

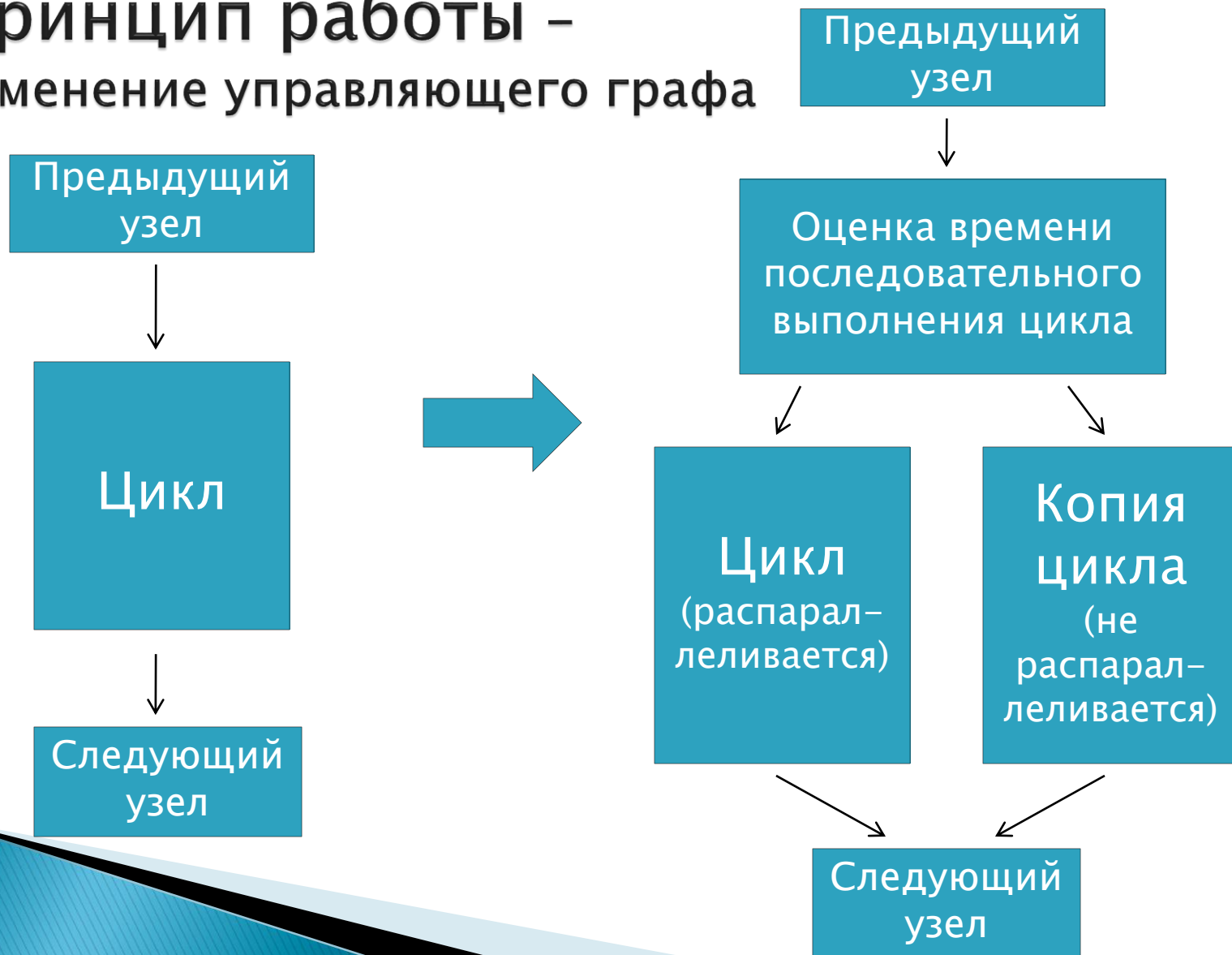
Динамическая проверка эффективности применения автоматического распараллеливания

## Проблематика и цель

- ▶ Существуют циклы, при распараллеливании которых потери гораздо больше, чем прирост производительности
- ▶ Статический метод оценки эффективности не всегда применим
- ▶ Необходимо реализовать алгоритм, позволяющий динамически определять эффективность применения автоматического распараллеливания

Динамическая проверка эффективности применения автоматического распараллеливания

## Принцип работы – изменение управляющего графа



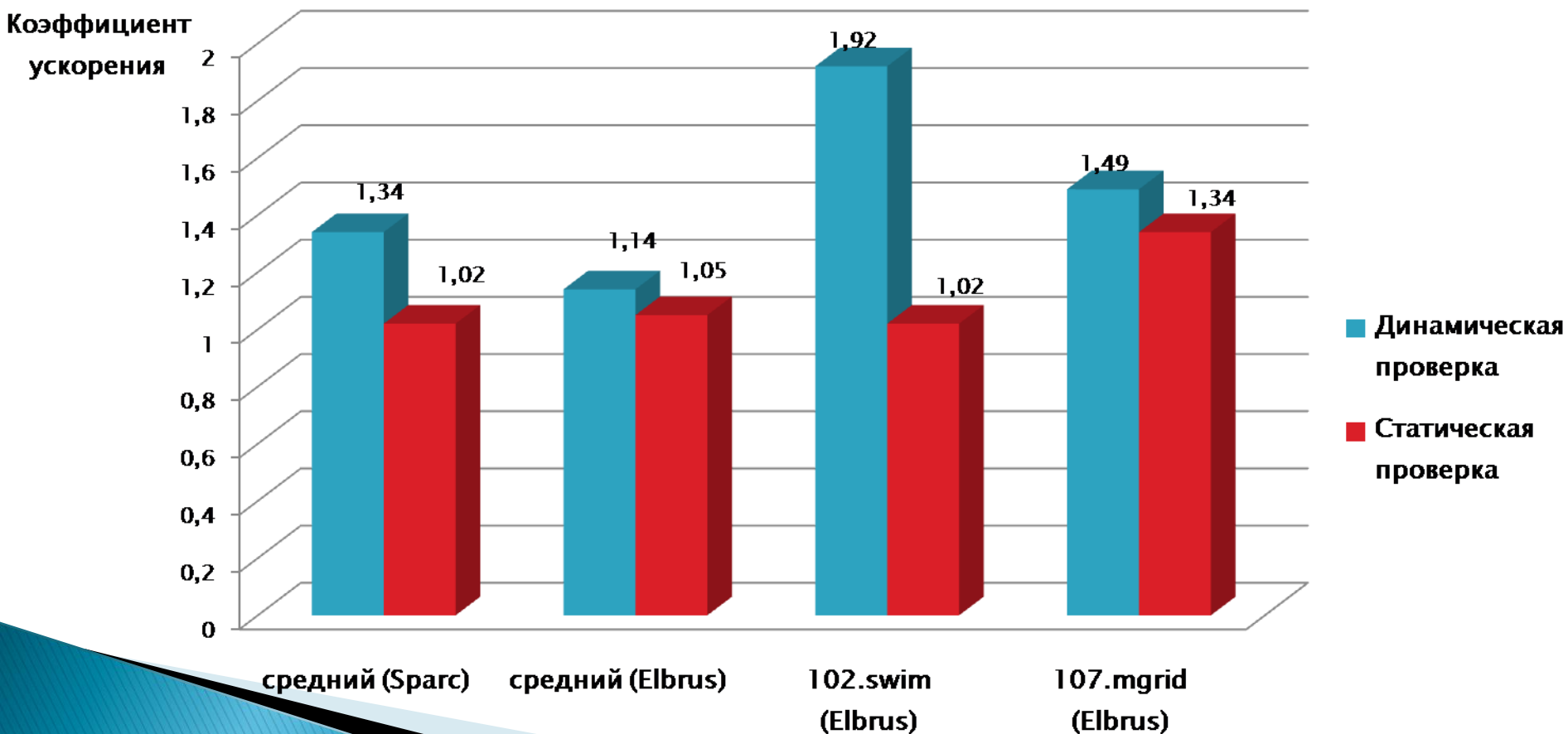
# Динамическая проверка эффективности применения автоматического распараллеливания

## Статическая и динамическая проверки

- ▶ Статическая проверка
  - В цикле много операций (эвристика) – распараллеливаем
- ▶ Динамическая проверка
  - На основе планирования операций (оценка):
    - Суммируем время (в тактах) выполнения всех операций в цикле;
  - На основе пропускной способности памяти (оценка):
    - Считаем суммарное время обращений цикла в память;
  - Выбираем большее из времен
  - Сравниваем с константой, полученной эмпирически
    - Время большое => переходим на параллельную версию цикла

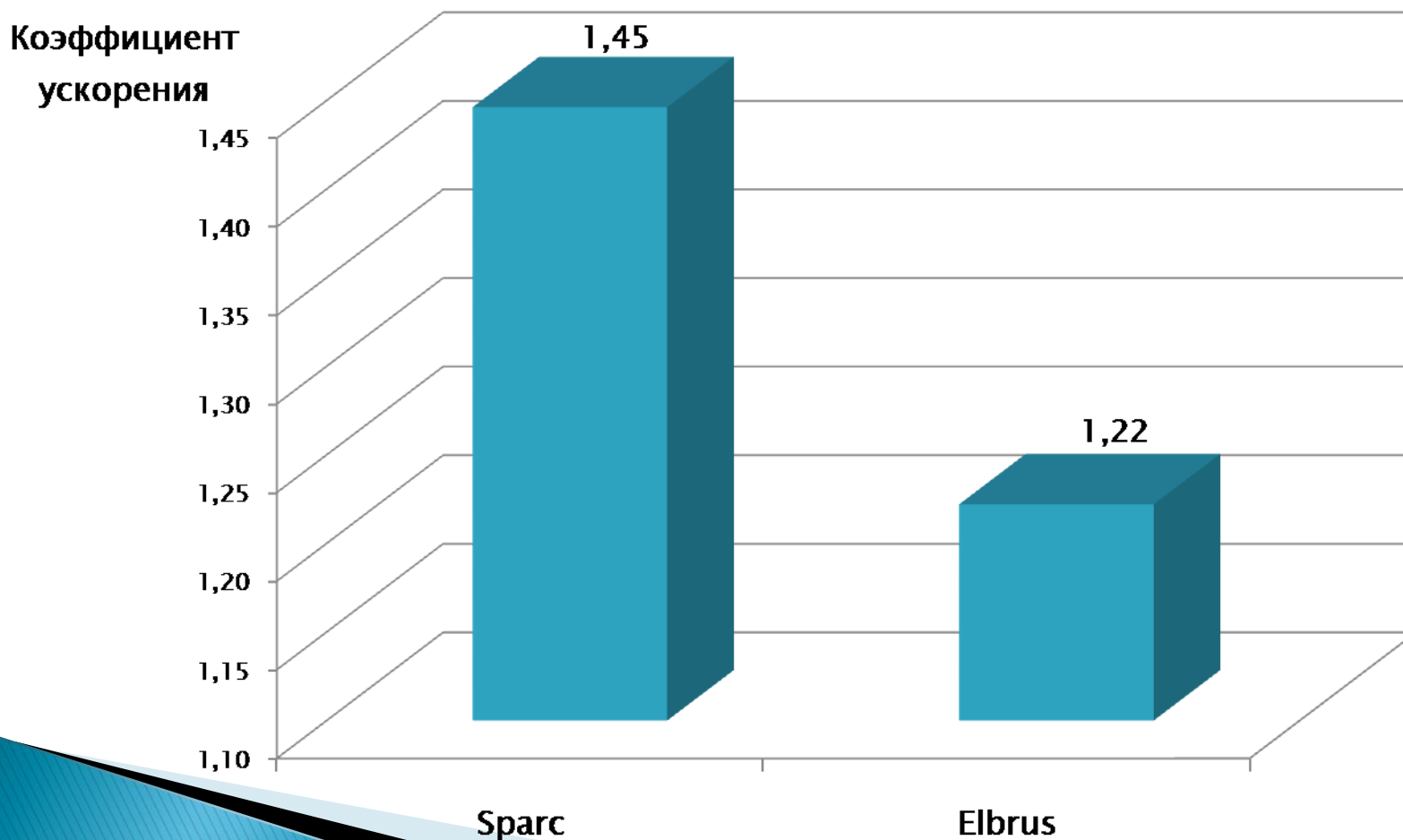
Динамическая проверка эффективности применения автоматического распараллеливания

# Результаты тестирования на пакете Spres95 на машинах Sparc и Elbrus



Динамическая проверка эффективности применения  
автоматического распараллеливания

# Результаты распараллеливания пакета GEMM(STDC) библиотеки EML





# Проблематика и цель

- Существует определенный класс рекурсивных алгоритмов

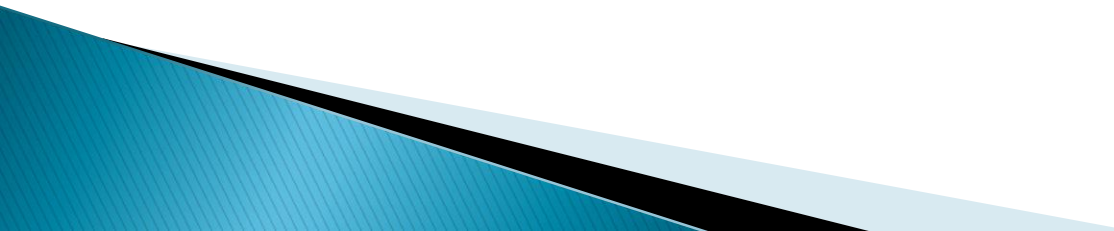
```
▶ void quickSort( int low, int high) {  
▶   int p, lastsmall, i;  
▶   if ( low < high) {  
▶     ...  
▶     quickSort( low, lastsmall); ← Thread1  
▶     quickSort( lastsmall+1, high); ← Thread2  
▶     sync;  
▶   }  
▶   return;  
▶ }
```

Необходимо реализовать анализ диапазонов обращения в память, с помощью которого можно было бы определить независимость вызовов

### Реализация

- ▶ Для процедуры строим диапазоны:
  - Def — множество адресов, по которым процедура записывает данные
  - Use — множество адресов, по которым процедура читает данные
- ▶ Пусть есть два рекурсивных вызова. Подставляя их параметры в найденные диапазоны, находим диапазоны, в которые обращаются вызовы:  $def_1, use_1; def_2, use_2$
- ▶ Критерий независимости
  - Для того чтобы два вызова были независимы, необходимо и достаточно:
    - $def_1 \cap def_2 = \emptyset$ ;
    - $def_1 \cap use_2 = \emptyset$ ;
    - $def_2 \cap use_1 = \emptyset$ .

## Результаты

- ▶ Реализован анализ для поиска диапазонов обращения процедуры в память
  - ▶ Правильность анализа проверена на алгоритме быстрой сортировки
  - ▶ Кроме того, данный анализ может быть использован как анализ общего назначения, т.к. может находить диапазоны не только процедур, но и любых операций, обращающихся в память
- 

Техника трансформации циклов на основе аффинных преобразований

## Проблематика и цели

- ▶ Приведение циклов к виду, в котором к ним может быть применено автоматическое распараллеливание
- ▶ Необходимо изучить модель аффинных преобразований и возможность ее применения для автоматического распараллеливания

# Техника трансформации циклов на основе аффинных преобразований

## Основные понятия

- Аффинная функция:

$$f(x) = c_0 + c_1x_1 + \dots + c_nx_n$$

- Аффинные обращения к массивам:  $X[i+1] = X[f(i)]$
- Инструкция – инструкция исходного языка программы
- В каждой инструкции содержится несколько обращений в память
- Итерационное пространство – множество значений, которые могут принимать индексные переменные гнезда циклов

# Техника трансформации циклов на основе аффинных преобразований

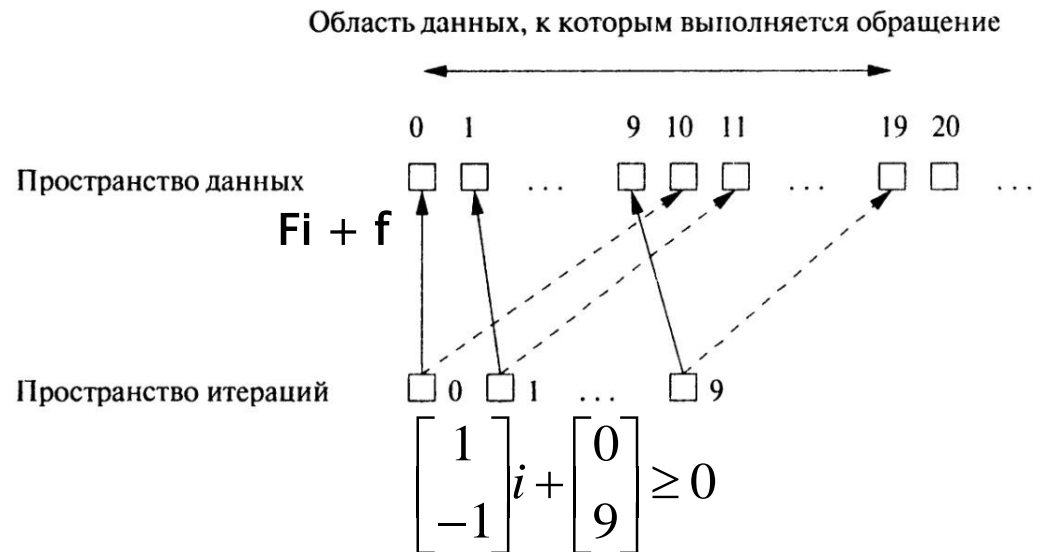
## ОСНОВНЫЕ ПОНЯТИЯ

- Математически итерационное пространство гнезда циклов глубиной  $d$  может быть представлено как
  - $\{i \in \mathbb{Z}^d \mid \mathbf{V}i + \mathbf{b} \geq \mathbf{0}\}$ , где
    - $\mathbf{V}$  – матрица  $2d \times d$ ,  $\mathbf{b}$  – вектор длины  $2d$
- Аффинные функции обращения к данным отображают вектор  $i$  в границы

$\mathbf{V}i + \mathbf{b} \geq \mathbf{0}$   
на элемент массива  
 $\mathbf{F}i + \mathbf{f}$ .

Пример:

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



Техника трансформации циклов на основе аффинных преобразований

## Поиск параллельности, не требующей синхронизации

- Разбиваем многогранник пространства итераций на подпространства, каждое подпространство исполняем на отдельном треде
- Инструкции, между которыми есть зависимость, относим к одному подпространству
- Независимые инструкции принадлежат разным подпространствам
- $p = C_i + c$  – аффинное разбиение, где  $p$  – идентификатор треда

Техника трансформации циклов на основе аффинных преобразований

## Поиск параллельности, не требующей синхронизации

- ▶ Составляем систему уравнений
  - Для всех пар зависимых обращений, т.е. таких что
    - $F_1 i_1 + f_1 = F_2 i_2 + f_2,$
  - при этом
    - $V_1 i_1 + b_1 \geq 0,$
    - $V_2 i_2 + b_2 \geq 0,$
  - выполняется соотношение  $C_1 i_1 + c_1 = C_2 i_2 + c_2.$
- ▶ Решая ее с помощью симплекс-метода, находим разбиения



# Техника трансформации циклов на основе аффинных преобразований

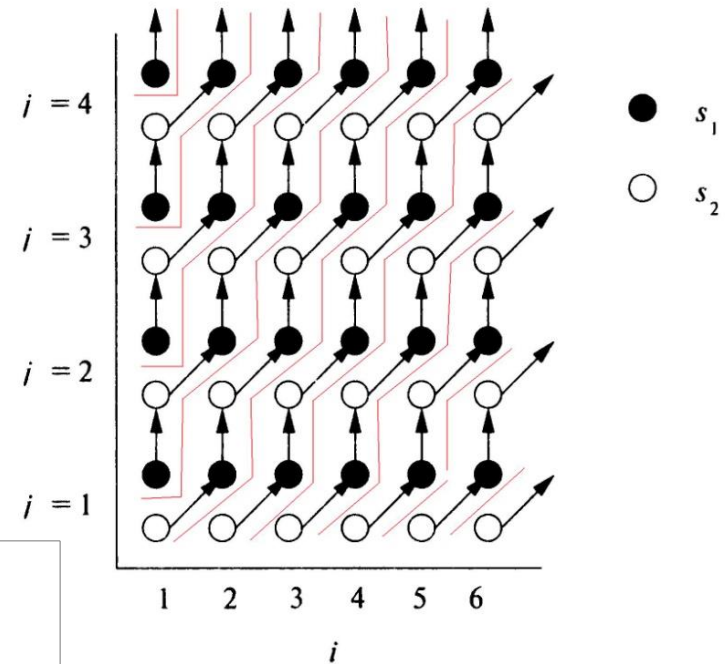
## Поиск параллельности, не требующей синхронизации

- Пример:

```
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  
  }
```

### После преобразования:

```
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  
    }
```



Техника трансформации циклов на основе аффинных преобразований

## Поиск параллельности с синхронизацией

- Вместо разбиения на подпространства строим для каждой инструкции «расписание»
- «Расписание» – функция, которая каждой инструкции ставит в соответствие момент времени, когда она должна выполняться
- Все инструкции, для которых значения «расписаний» совпадают, могут быть выполнены параллельно:

```
for( t = 0; t < L; t++){  
    for( i = 0; i < N; i++) <- parallel loop  
    ...  
    barrier;  
}
```

Техника трансформации циклов на основе аффинных преобразований

## Поиск параллельности с синхронизацией

- **Пример: 1-d Jakobí**

```
for (t = 0; t < T; t++) {  
    for (i = 1; i < N - 1; i++)  
        B[i] = (A[i-1] + A[i] + A[i+1])/3; //S1  
    for (j = 1; j < N - 1; j++)  
        A[j] = B[j]; //S2  
}
```

- **После преобразования**

```
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] =  
                (A[q - 4 * t] + A[q - 4 * t + 1] + A[q - 4 * t + 2])/3; //S1  
        if ((q - 4 * t - 1 >= 1) && (q - 4 * t - 1 < N))  
            A[q - 4 * t - 1] = B[q - 4 * t - 1]; //S2  
    }  
    barrier;  
}
```

# Техника трансформации циклов на основе аффинных преобразований

## Результаты

- ▶ Исследована возможность применения техники аффинных преобразований для автоматического распараллеливания циклов
- ▶ Системы уравнений, применяемые в этой технике, могут быть составлены и решены средствами оптимизирующего компилятора

# Результаты

- ▶ Реализован алгоритм динамической проверки эффективности применения автоматического распараллеливания, проведена его интеграция в оптимизирующий компилятор и тестирование. Он также позволил эффективно автоматически распараллелить библиотеку EML
- ▶ Реализован анализ диапазонов обращения процедур в память для реализации распараллеливания рекурсивных процедур
- ▶ Проведено исследование техники аффинных преобразований и возможности ее применения для автоматического распараллеливания