ТАЙМИРУЮЩИЕ ФУНКЦИИ

(дополнение к разделу 1.3)

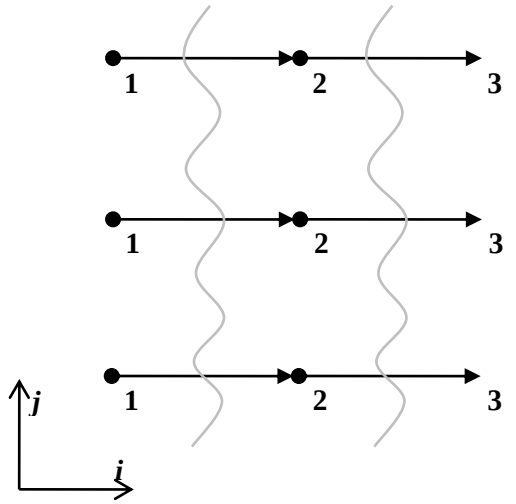
Определение таймирующей функции («timing function») дано в разделе 1.3.

Значения таймирующей функции: операциям $S_\beta(J)$ приписаны некоторые числа («время» выполнения операций), упорядоченные в соответствии с зависимостями.

В англоязычной литературе теперь принят термин «scheduling function». В русскоязычной встречается название (Воеводин В.В.) «развёртка графа алгоритма»: вещественный функционал, определённый на вершинах графа алгоритма, называется развёрткой графа, если он не убывает вдоль дуг графа.

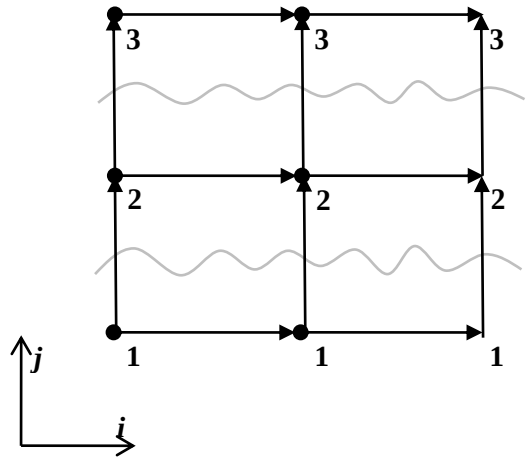
Использование таймирующих функций

1. Любая (расщепляющая, строгая, «нестрогая») таймирующая функция задаёт информационные разрезы алгоритма, т.е. задает разбиение алгоритма на части, которые можно выполнять последовательно друг за другом в порядке возрастания значений функции.

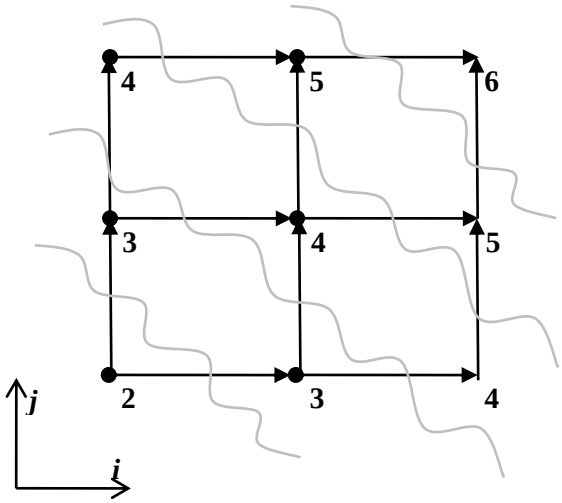


$t(i,j)=i$

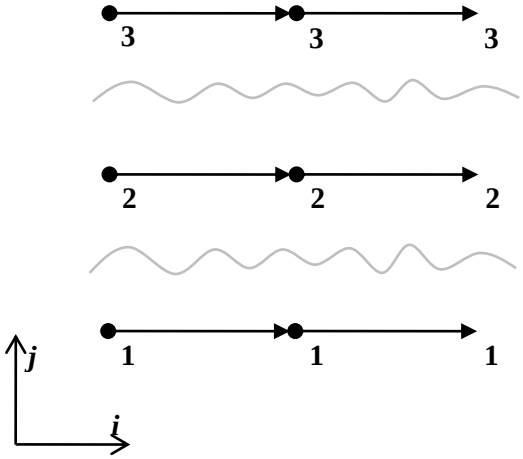
Здесь и далее одному значению функции соответствует одно множество операций алгоритма.



$t(i,j)=j$

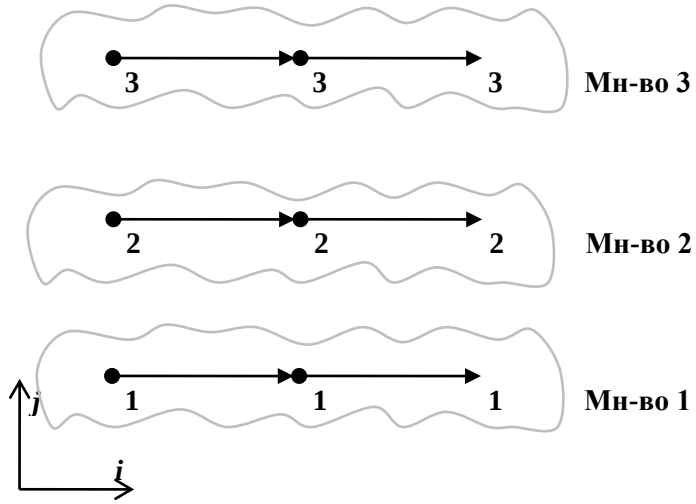


$t(i,j)=i+j$



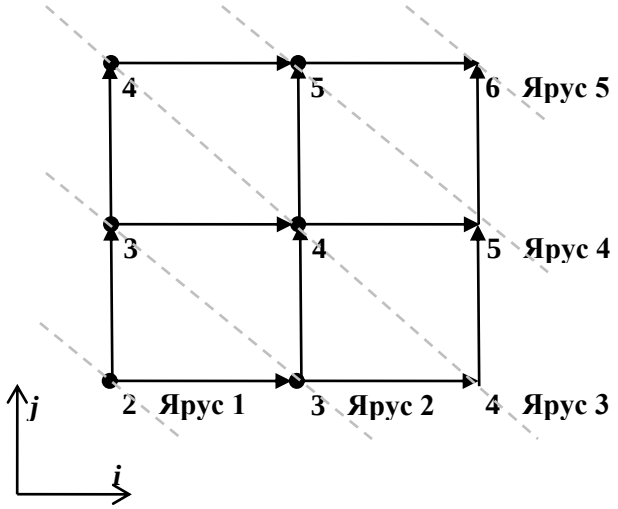
$t(i,j)=j$

2. Расщепляющие таймирующие функции задают параллельные множества операций алгоритма (разбиение операций на группы, которые могут быть выполнены независимо друг от друга).



$t(i,j)=j$

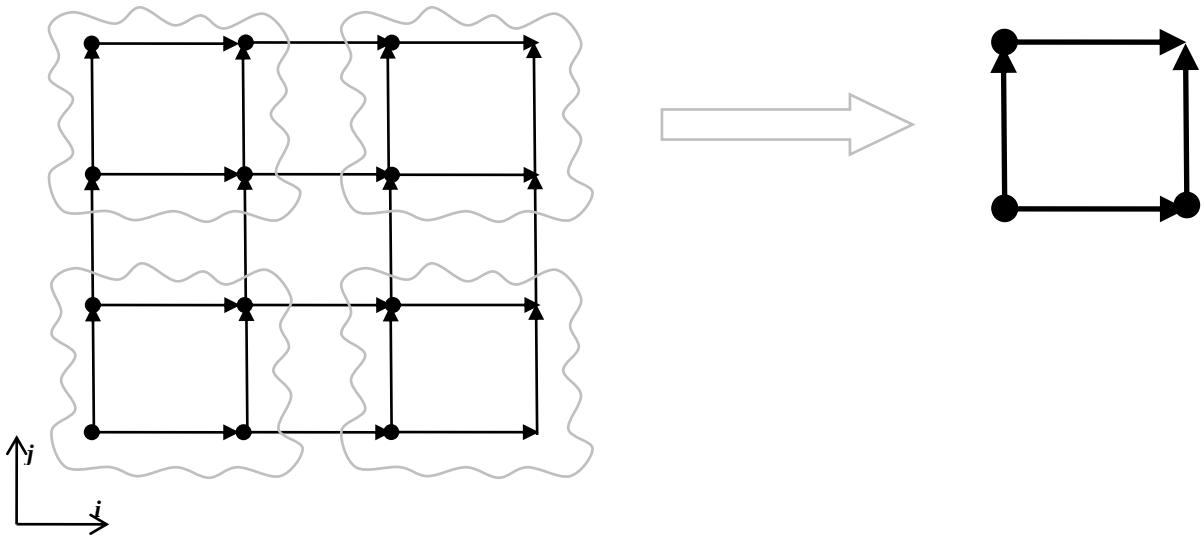
3. Строгие таймирующие функции задают параллельные формы алгоритма (разбиение операций на группы, в которых операции могут быть выполнены независимо).



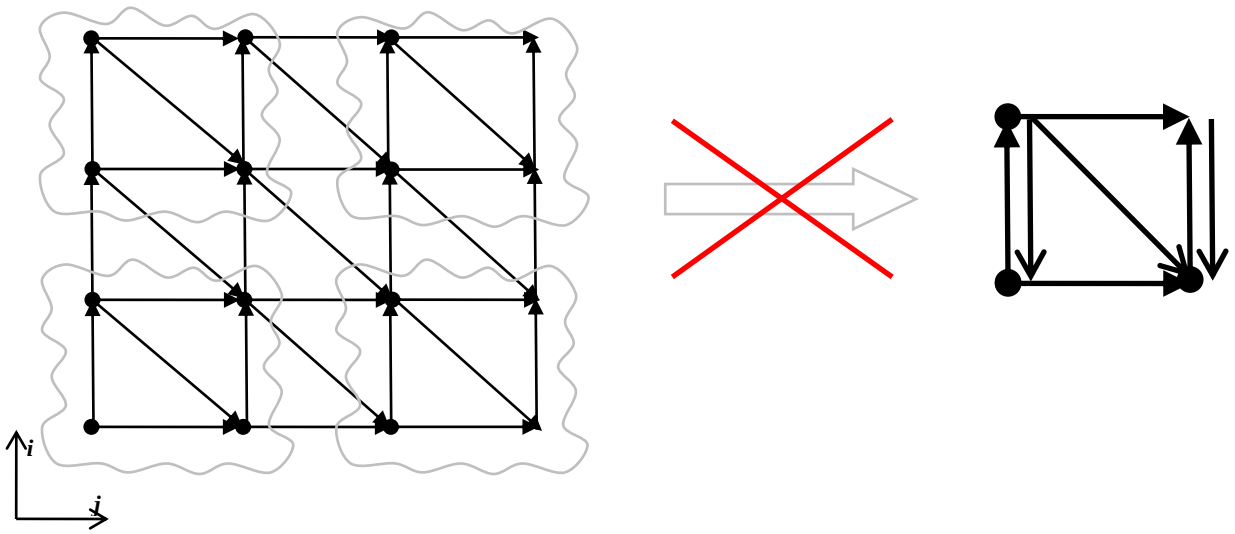
$t(i,j)=i+j$

Параллелизм, задаваемый таймирующими функциями, называется скошенным. Частным случаем скошенного параллелизма является координатный параллелизм. Координатный параллелизм определяется параллельным циклом исходного гнезда циклов.

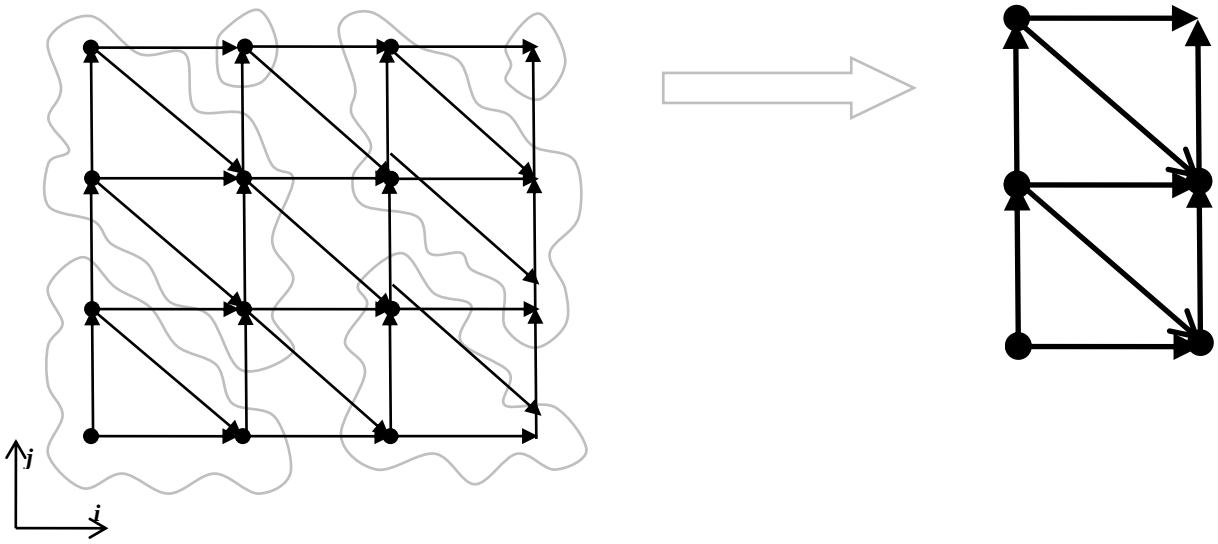
4. Независимые друг от друга таймирующие функции задают семейства информационных разрезов алгоритма и поэтому могут быть использованы для увеличения зернистости вычислений (т.е. для разбиения операций алгоритма на макрооперации-итайлы). В следующем примере можно использовать независимые таймирующие функции $t(i,j)=i$ и $t(i,j)=j$:



Обратных связей между тайлами быть не должно. Например, такое разбиение недопустимо (используется функция $t(i,j)=j$, не являющаяся таймирующей):



Можно произвести разбиение следующим образом (используются две независимые таймирующие функции):



5. Векторные таймирующие функции задают преобразования циклов. Преобразования циклов применяются для получения параллельных циклов (см. файл «Параллельность циклов»), а также для улучшения структуры последовательной программы (например, для получения макроопераций, для сжатия массивов).

Приведем пример. Рассмотрим двумерный цикл:

```
do  i = 1, N
    do  j = 1, N
        S1:  a(i,j) = a(i–1,j) + a(i,j–1)
    enddo
enddo
```

Применим преобразование $i'=i+j,\ j'=j$ алгоритма, задаваемое векторной таймирующей функцией $\left(t_1^{S_1}(i,j), t_2^{S_1}(i,j)\right)=\left(i+j, j\right)$. Получим (см. пример в разделе 1.4)

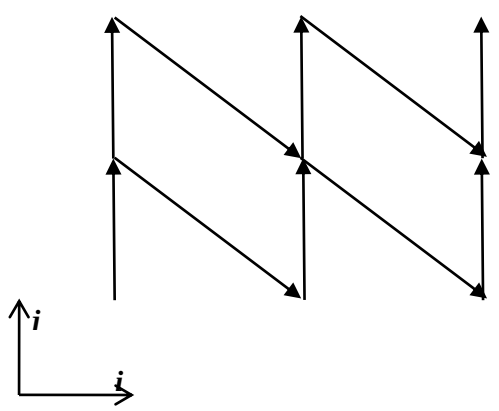
```
do  i' = 2, 2N
    do   j' = max(1, i'–N), min(i'–1, N)
         S1:  a(i' – j' , j' ) = a(i' – j' –1, j' ) + a(i' – j' , j' –1)
    enddo
enddo
```

Пример векторной таймирующей функции, вторая компонента которой не является таймирующей функцией

Таймирующая функции вида $t(J)=\tau J,\ \tau\in Z^n$, называется координатной таймирующей функцией, если вектор τ содержит ровно одну отличную от нуля компоненту. Можно гарантировать наличие стольких координатных таймирующих функций, сколько координат векторов, характеризующих зависимости, оказываются неотрицательными. Действительно, если ζ-я компонента вектора τ равна 1, остальные компоненты равны 0, компонента с номером ζ любого вектора зависимостей $J–I$ неотрицательна, то функции $t(J)=\tau J$ является таймирующей: $t(J)–t(I)=t(j_1,...,j_n)–t(i_1,...,i_n)=j_\zeta–i_\zeta\geq 0$.

Пусть задана векторная таймирующая функция $\left(t_1,t_2,...,t_n\right)$, $t_i(J)=J$. Функция t_1 задает порядок выполнения операций в соответствии с итерациями самого внешнего цикла и поэтому всегда является таймирующей. Координатные функции $t_i(J)$, $i\geq 2$, не обязательно являются таймирующими. Они упорядочивают операции алгоритма в соответствии с внутренними циклами при фиксированных значениях внешних циклов. Приведем пример, в котором t_2 не является таймирующей функцией:

```
do i = 1, N
    do j = 1, N
        a(i,j) = a(i,j–1) + a(i–1,j+1)
    enddo
enddo
```



$$\varphi^1=(0,\ 1),$$
$$\varphi'^2=(1,\ -1),$$
$$\bar{t}\,(i,\ j)=(t_1(i,j),t_2(i,j))=(i,j).$$

Здесь $t_2(i,j)$ не является таймирующей функцией (не выполняется условие сохранения зависимостей).