Лекция 2. Распараллеливающие преобразования компиляторов

Инструменты разработки быстрых программ

20 ноября 2017 г.

Вхождения переменных

Определения

Вхождение переменой: пара: переменная + место в тексте программы (алгоритме), где она встречается. Соответствует обращению к некоторой ячейке памяти.

Генератор (out): вхождение переменной, которому соответствует запись в ячейку памяти.

Использование (in): вхождение переменной, которому соответствует чтение из ячейки памяти.

Примеры генераторов и использований

Пример

```
  fA = fB + fC;
// 1    2    3
```

Примеры генераторов и использований

Пример

```
  fA = fB + fC;
// 1    2    3
```

Пример

```
  afA[i] = afB[i] + afC[i - 1];
//  1 2      3 4      5 6
```

Информационная зависимость

Определение

Два вхождения порождают информационную зависимость, если (при некоторых допустимых значения индексных выражений в случае массивов) они обращаются к одной и той же ячейке памяти.

Примеры информационных зависимостей

Пример

```
for (i = 1; i < 10; ++ i)
  afA[i] = afA[i - 1] + 4;
//  1         2
```

Примеры информационных зависимостей

Пример

```
for (i = 1; i < 10; ++ i)
  afA[i] = afA[i - 1] + 4;
//  1         2
```

Пример

```
istr >> N;
for (i = 10; i < N; ++ i)
  afA[i] = afA[i - 10] + 4;
//  1         2
```

Виды информационных зависимостей

По способу обращения к ячейке памяти

- Циклически независимая.
- Циклически порождённая.

По порядку чтения/записи

Вид зависимости	Вхождение раньше	Вхождение позже
Истинная (out-in)	генератор	использование
Антизависимость (in-out)	использование	генератор
Выходная (out-out)	генератор	генератор
Входная (in-in)	использование	использование

Виды информационных зависимостей

По способу обращения к ячейке памяти

- Циклически независимая.
- Циклически порождённая.

По порядку чтения/записи

Вид зависимости	Вхождение раньше	Вхождение позже
Истинная (out-in)	генератор	использование
Антизависимость (in-out)	использование	генератор
Выходная (out-out)	генератор	генератор
Входная (in-in)	использование	использование

Примеры информационных зависимостей

Пример (out-in, in-out, out-out)

```
int main()
{
  double dA, dB;
  dA = 2;          // (1)
  dB = 4 - dA;     // (2)
  dA = 8;          // (3)
}
```

Пример (in-in)

```
int main()
{
  double dA, dB, dC;
  dA = 2;
  dB = 4 - dA;     // (2)
  dC = 4 + dA;     // (3)
}
```

Примеры информационных зависимостей (продолжение)

Пример (циклически независимая)

```
int main()
{
  int i;
  double adA[1000];
  // ...
  for (i = 0; i < 1000; ++ i)
  {
    adA[i] = i;              // (1)
    adA[i] = 2 + adA[i];     // (2)
  }
}
```

Пример (циклически порождённые)

```
int main()
{
  int i;
  double adA[1000];
  // ...
  for (i = 1; i < 1000; ++ i)
  {
    adA[i] = i;              // (1)
    adA[i] = 2 + adA[i - 1];// (2)
  }
}
```

Примеры информационных зависимостей (продолжение)

Пример (раскрутка цикла)

```
int main()
{
  int i;
  double adA[1000];
  // ...
  adA[1] = 1;              // (1.1)
  adA[1] = 2 + adA[0];     // (2.1)
  adA[2] = 2;              // (1.2)
  adA[2] = 2 + adA[1];     // (2.2)
  // ...
}
```

Примеры информационных зависимостей (продолжение)

Пример (out-in)

```
for (i = 1; i < 1000; ++ i)
  adA[i] = 2 + adA[i - 1];
```

Пример (in-out)

```
for (i = 0; i < 999; ++ i)
  adA[i] = 2 + adA[i + 1];
```

Пример (out-out)

```
for (i = 0; i < 999; ++ i)
{
  adA[i + 1] = 2 * i;
  adA[i]     = 3 * i;
}
```

Пример (in-in)

```
for (i = 1; i < 999; ++ i)
  adB[i] =
    adA[i - 1] + adA[i + 1];
```

Примеры информационных зависимостей (продолжение)

Пример (out-in)

```
for (i = 1; i < 1000; ++ i)
  adA[i] = 2 + adA[i - 1];
```

Пример (in-out)

```
for (i = 0; i < 999; ++ i)
  adA[i] = 2 + adA[i + 1];
```

Пример (out-out)

```
for (i = 0; i < 999; ++ i)
{
  adA[i + 1] = 2 * i;
  adA[i]     = 3 * i;
}
```

Пример (in-in)

```
for (i = 1; i < 999; ++ i)
  adB[i] =
    adA[i - 1] + adA[i + 1];
```

Примеры информационных зависимостей (продолжение)

Пример (out-in)

```
for (i = 1; i < 1000; ++ i)
  adA[i] = 2 + adA[i - 1];
```

Пример (in-out)

```
for (i = 0; i < 999; ++ i)
  adA[i] = 2 + adA[i + 1];
```

Пример (out-out)

```
for (i = 0; i < 999; ++ i)
{
  adA[i + 1] = 2 * i;
  adA[i]     = 3 * i;
}
```

Пример (in-in)

```
for (i = 1; i < 999; ++ i)
  adB[i] =
    adA[i - 1] + adA[i + 1];
```

Примеры информационных зависимостей (продолжение)

Пример (out-in)

```
for (i = 1; i < 1000; ++ i)
  adA[i] = 2 + adA[i - 1];
```

Пример (in-out)

```
for (i = 0; i < 999; ++ i)
  adA[i] = 2 + adA[i + 1];
```

Пример (out-out)

```
for (i = 0; i < 999; ++ i)
{
  adA[i + 1] = 2 * i;
  adA[i]     = 3 * i;
}
```

Пример (in-in)

```
for (i = 1; i < 999; ++ i)
  adB[i] =
    adA[i - 1] + adA[i + 1];
```

Примеры информационных зависимостей (окончание)

Пример (невозможность определения зависимости)

```
int main()
{
  int i, k;
  double adA[1000];
  // ...
  k = f();
  for (i = max(0, k); i < min(1000, 1000 + k); ++ i)
    adA[i] = adA[i - k];
  // ...
}
```

Примеры информационных зависимостей (окончание)

Пример (невозможность определения зависимости)

```
int main()
{
  int i, k;
  double adA[1000];
  // ...
  k = f();
  for (i = max(0, k); i < min(1000, 1000 + k); i += 2)
    adA[i] = adA[i - k];
  // ...
}
```

Использование информационных зависимостей

Утверждение

- *Если в цикле есть только операторы присваивания и нет информационных зависимостей между его итерациями, то их можно выполнять независимо на асинхронных архитектурах (MIMD).*
- *На архитектурах с разделяемой памятью допускаются также входные зависимости (in-in).*
- *На архитектурах с распределённой памятью проблема входных зависимостей между различными итерациями цикла может быть решена предварительной рассылкой данных вычислительным узлам.*

Использование (окончание)

Допустимые виды информационных зависимостей

MIMD — допускаются входные зависимости (возможно, нужна рассылка данных).

SIMD — допускаются зависимости от вхождений, выполняемых раньше ко вхождениям, выполняемым позже:

- входные зависимости;
- антизависимости;
- выходные зависимости.

Алгоритм умножения матриц

Пример (умножение матриц)

входные данные: $A, B \in M_{n\times n}(\mathbb{R})$
выходные данные: $C \in M_{n\times n}(\mathbb{R})$
начало
  для каждого $i \in [1,n]_{\mathbb{N}}$ **выполнять**
    для каждого $j \in [1,n]_{\mathbb{N}}$ **выполнять**
      $c_{ij} \leftarrow 0$
      для каждого $k \in [1,n]_{\mathbb{N}}$ **выполнять**
        $c_{ij} \leftarrow c_{ij} + a_{ik} \quad b_{kj}$

Зависимости

Пер.	Цикл	Тип
c_{ij}	–	out-in
c_{ij}	–	out-out
c_{ij}	–	in-out
c_{ij}	k	out-in
c_{ij}	k	out-out
c_{ij}	k	in-in
a_{ik}	j	in-in
b_{kj}	i	in-in

Алгоритм умножения матриц

Пример (умножение матриц)

входные данные: $A, B \in M_{n\times n}(\mathbb{R})$
выходные данные: $C \in M_{n\times n}(\mathbb{R})$
начало
  для каждого $i \in [1,n]_{\mathbb{N}}$ **выполнять**
    для каждого $j \in [1,n]_{\mathbb{N}}$ **выполнять**
      $c_{ij} \leftarrow 0$
      для каждого $k \in [1,n]_{\mathbb{N}}$ **выполнять**
        $c_{ij} \leftarrow c_{ij} + a_{ik}\ \ b_{kj}$

Зависимости

Пер.	Цикл	Тип
c_{ij}	–	out-in
c_{ij}	–	out-out
c_{ij}	–	in-out
c_{ij}	k	out-in
c_{ij}	k	out-out
c_{ij}	k	in-in
a_{ik}	j	in-in
b_{kj}	i	in-in

Алгоритм умножения матриц

Пример (умножение матриц)

входные данные: $A, B \in M_{n\times n}(\mathbb{R})$
выходные данные: $C \in M_{n\times n}(\mathbb{R})$
начало
  для каждого $i \in [1,n]_{\mathbb{N}}$ **выполнять**
    для каждого $j \in [1,n]_{\mathbb{N}}$ **выполнять**
      $c_{ij} \leftarrow 0$
      для каждого $k \in [1,n]_{\mathbb{N}}$ **выполнять**
        $c_{ij} \leftarrow c_{ij} + a_{ik}\ \ b_{kj}$

Зависимости

Пер.	Цикл	Тип
c_{ij}	–	out-in
c_{ij}	–	out-out
c_{ij}	–	in-out
c_{ij}	k	out-in
c_{ij}	k	out-out
c_{ij}	k	in-in
a_{ik}	j	in-in
b_{kj}	i	in-in

Алгоритм умножения матриц

Пример (умножение матриц)

входные данные: $A, B \in M_{n\times n}(\mathbb{R})$
выходные данные: $C \in M_{n\times n}(\mathbb{R})$
начало
  для каждого $i \in [1,n]_{\mathbb{N}}$ **выполнять**
    для каждого $j \in [1,n]_{\mathbb{N}}$ **выполнять**
      $c_{ij} \leftarrow 0$
      для каждого $k \in [1,n]_{\mathbb{N}}$ **выполнять**
        $c_{ij} \leftarrow c_{ij} + a_{ik}\ \ b_{kj}$

Зависимости

Пер.	Цикл	Тип
c_{ij}	–	out-in
c_{ij}	–	out-out
c_{ij}	–	in-out
c_{ij}	k	out-in
c_{ij}	k	out-out
c_{ij}	k	in-in
a_{ik}	j	in-in
b_{kj}	i	in-in

Алгоритм умножения матриц

Пример (умножение матриц)

входные данные: $A, B \in M_{n\times n}(\mathbb{R})$
выходные данные: $C \in M_{n\times n}(\mathbb{R})$
начало
  для каждого $i \in [1,n]_{\mathbb{N}}$ **выполнять**
    для каждого $j \in [1,n]_{\mathbb{N}}$ **выполнять**
      $c_{ij} \leftarrow 0$
      для каждого $k \in [1,n]_{\mathbb{N}}$ **выполнять**
        $c_{ij} \leftarrow c_{ij} + a_{ik}\ \ b_{kj}$

Зависимости

Пер.	Цикл	Тип
c_{ij}	–	out-in
c_{ij}	–	out-out
c_{ij}	–	in-out
c_{ij}	k	out-in
c_{ij}	k	out-out
c_{ij}	k	in-in
a_{ik}	j	in-in
b_{kj}	i	in-in

Алгоритм умножения матриц

Пример (умножение матриц)

входные данные: $A, B \in M_{n\times n}(\mathbb{R})$
выходные данные: $C \in M_{n\times n}(\mathbb{R})$
начало
  для каждого $i \in [1,n]_{\mathbb{N}}$ **выполнять**
    для каждого $j \in [1,n]_{\mathbb{N}}$ **выполнять**
      $c_{ij} \leftarrow 0$
      для каждого $k \in [1,n]_{\mathbb{N}}$ **выполнять**
        $c_{ij} \leftarrow c_{ij} + a_{ik}\ \ b_{kj}$

Зависимости

Пер.	Цикл	Тип
c_{ij}	–	out-in
c_{ij}	–	out-out
c_{ij}	–	in-out
c_{ij}	k	out-in
c_{ij}	k	out-out
c_{ij}	k	in-in
a_{ik}	j	in-in
b_{kj}	i	in-in

Алгоритм умножения матриц

Пример (умножение матриц)

входные данные: $A, B \in M_{n\times n}(\mathbb{R})$
выходные данные: $C \in M_{n\times n}(\mathbb{R})$
начало
  для каждого $i \in [1,n]_{\mathbb{N}}$ **выполнять**
    для каждого $j \in [1,n]_{\mathbb{N}}$ **выполнять**
      $c_{ij} \leftarrow 0$
      для каждого $k \in [1,n]_{\mathbb{N}}$ **выполнять**
        $c_{ij} \leftarrow c_{ij} + a_{ik}\ \ b_{kj}$

Зависимости

Пер.	Цикл	Тип
c_{ij}	–	out-in
c_{ij}	–	out-out
c_{ij}	–	in-out
c_{ij}	k	out-in
c_{ij}	k	out-out
c_{ij}	k	in-in
a_{ik}	j	in-in
b_{kj}	i	in-in

Алгоритм умножения матриц

Пример (умножение матриц)

входные данные: $A, B \in M_{n\times n}(\mathbb{R})$
выходные данные: $C \in M_{n\times n}(\mathbb{R})$
начало
  для каждого $i \in [1,n]_{\mathbb{N}}$ **выполнять**
    для каждого $j \in [1,n]_{\mathbb{N}}$ **выполнять**
      $c_{ij} \leftarrow 0$
      для каждого $k \in [1,n]_{\mathbb{N}}$ **выполнять**
        $c_{ij} \leftarrow c_{ij} + a_{ik}\ b_{kj}$

Зависимости

Пер.	Цикл	Тип
c_{ij}	–	out-in
c_{ij}	–	out-out
c_{ij}	–	in-out
c_{ij}	k	out-in
c_{ij}	k	out-out
c_{ij}	k	in-in
a_{ik}	j	in-in
b_{kj}	i	in-in

Алгоритм умножения матриц

Пример (умножение матриц)

входные данные: $A, B \in M_{n\times n}(\mathbb{R})$
выходные данные: $C \in M_{n\times n}(\mathbb{R})$
начало
  для каждого $i \in [1,n]_{\mathbb{N}}$ **выполнять**
    для каждого $j \in [1,n]_{\mathbb{N}}$ **выполнять**
      $c_{ij} \leftarrow 0$
      для каждого $k \in [1,n]_{\mathbb{N}}$ **выполнять**
        $c_{ij} \leftarrow c_{ij} + a_{ik}\ b_{kj}$

Зависимости

Пер.	Цикл	Тип
c_{ij}	–	out-in
c_{ij}	–	out-out
c_{ij}	–	in-out
c_{ij}	k	out-in
c_{ij}	k	out-out
c_{ij}	k	in-in
a_{ik}	j	in-in
b_{kj}	i	in-in

Алгоритм умножения матриц

Пример (умножение матриц)

входные данные: $A, B \in M_{n\times n}(\mathbb{R})$
выходные данные: $C \in M_{n\times n}(\mathbb{R})$
начало
  для каждого $i \in [1,n]_{\mathbb{N}}$ **выполнять**
    для каждого $j \in [1,n]_{\mathbb{N}}$ **выполнять**
      $c_{ij} \leftarrow 0$
      для каждого $k \in [1,n]_{\mathbb{N}}$ **выполнять**
        $c_{ij} \leftarrow c_{ij} + a_{ik} b_{kj}$

Зависимости

Пер.	Цикл	Тип
c_{ij}	–	out-in
c_{ij}	–	out-out
c_{ij}	–	in-out
c_{ij}	k	out-in
c_{ij}	k	out-out
c_{ij}	k	in-in
a_{ik}	j	in-in
b_{kj}	i	in-in

Примеры допустимых зависимостей

Пример

```
for (i = 1; i < 1000; ++ i)
  adX[i] = adA[i] * adX[i - 1] +
    adB[i];
```

Пример

```
for (i = 1; i < 1000; ++ i)
{
  adX[i] = adA[i] * adY[i - 1] +
    adB[i];
  adY[i] = adD[i] / adX[i];
}
```

Примеры допустимых зависимостей

Пример

```
for (i = 1; i < 1000; ++ i)
  adX[i] = adA[i] * adX[i - 1] +
    adB[i];
```

Пример

```
for (i = 1; i < 1000; ++ i)
{
  adX[i] = adA[i] * adY[i - 1] +
    adB[i];
  adY[i] = adD[i] / adX[i];
}
```

Тесное гнездо циклов

Определение

Тесным гнездом циклов называется участок программы вида

для i_1 **от** $\underline{l_1}$ **до** $\overline{l_1}$ **выполнять**
  для i_2 **от** $\underline{l_2}$ **до** $\overline{l_2}$ **выполнять**
    ...
      для i_n **от** $\underline{l_n}$ **до** $\overline{l_n}$ **выполнять**
        Тело_цикла($i_1, i_2, \ldots, i_n$);
    ...

где $\underline{l_k}, \overline{l_k}$ — границы циклов — выражения, зависящие от счётчиков охватывающих циклов и от внешних переменных.

Пространство итераций

Вектор счётчиков цикла

$$i \stackrel{\text{def}}{=} (i_1, i_2, \ldots, i_n) \in \mathbb{Z}^n$$

Определение

Пространством итераций тесного гнезда циклов называется множество $D \subset \mathbb{Z}^n$ всех значений вектора счётчиков циклов, которые он принимает во время исполнения данного фрагмента программы.

Пространство итераций

Вектор счётчиков цикла

$$i \stackrel{\text{def}}{=} (i_1, i_2, \ldots, i_n) \in \mathbb{Z}^n$$

Определение

Пространством итераций тесного гнезда циклов называется множество $D \subset \mathbb{Z}^n$ всех значений вектора счётчиков циклов, которые он принимает во время исполнения данного фрагмента программы.

Пространство итераций

Пример

для i **от** 1 **до** $(n-1)$ **выполнять**
  для j **от** $(i+1)$ **до** n **выполнять**
    $\ldots$
    $\vec{F}_i \leftarrow \vec{F}_i + \ldots;$
    // u_1 v_1
    $\vec{F}_j \leftarrow \vec{F}_j - \ldots;$
    // u_2 v_2

Рис. 1: Пространство итераций цикла вычисления сил для 6 тел

Элементарный решётчатый граф

Определение

Пусть x — m-мерный массив, u, v — вхождения x в тесное гнездо циклов:

для каждого $i \in D$ **выполнять**

$x_{f(i)} \leftarrow \ldots;$
// u
$\ldots \leftarrow \ldots \quad x_{g(i)} \ldots;$
// v

где $f, g \colon \mathbb{Z}^n \to \mathbb{Z}^m$ — индексные выражения.

Пусть (u, v) связаны истинной информационной зависимостью.

Элементарный решётчатый граф (окончание)

Определение (окончание)

Элементарным решётчатым графом истинной зависимости, построенным по вхождениям (u, v) переменной x в гнезде циклов, называется граф, для которого:

1. Множеством вершин является пространство итераций тесного гнезда (D).
2. Дуга соединяет две вершины (i, j), если:
   - $i < j$.
   - $f(i) = g(j)$.
   - $\forall k \in D \mid (k < j) \wedge \big(f(k) = g(j)\big) \quad k \leqslant i$.

Раскрутка цикла

Пример (задача гравитации n тел)

$$(1,2)\begin{bmatrix} f_1 \leftarrow f_1 + \cdot \\ f_2 \leftarrow f_2 - \cdot \end{bmatrix}$$

$$(1,3)\begin{bmatrix} f_1 \leftarrow f_1 + \cdot \\ f_3 \leftarrow f_3 - \cdot \end{bmatrix} \quad (2,3)\begin{bmatrix} f_2 \leftarrow f_2 + \cdot \\ f_3 \leftarrow f_3 - \cdot \end{bmatrix}$$

$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$

$$(1,6)\begin{bmatrix} f_1 \leftarrow f_1 + \cdot \\ f_6 \leftarrow f_6 - \cdot \end{bmatrix} \quad (2,6)\begin{bmatrix} f_2 \leftarrow f_2 + \cdot \\ f_6 \leftarrow f_6 - \cdot \end{bmatrix} \quad \ldots \quad (5,6)\begin{bmatrix} f_5 \leftarrow f_5 + \cdot \\ f_6 \leftarrow f_6 - \cdot \end{bmatrix}$$

Элементарный решётчатый граф для цикла

Пример

для i **от** 1 **до** $(n-1)$ **выполнять**
  для j **от** $(i+1)$ **до** n **выполнять**
    $\ldots$
    $\vec{F}_i \leftarrow \vec{F}_i + \ldots;$
  // u_1 v_1
    $\vec{F}_j \leftarrow \vec{F}_j - \ldots;$
  // u_2 v_2

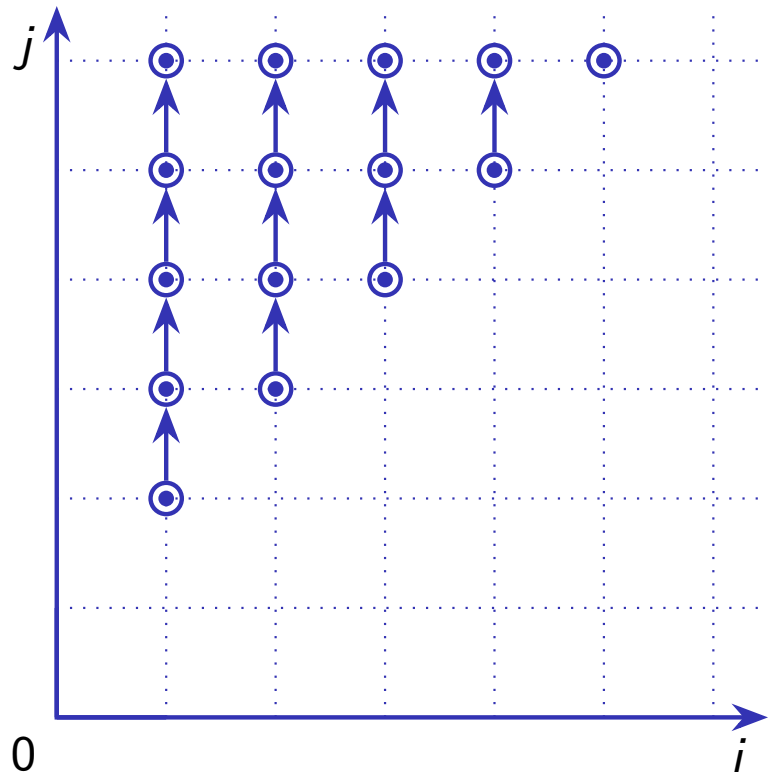

Рис. 2: Элементарный решётчатый граф вхождений (u_1, v_1)

Элементарный решётчатый граф для цикла

Пример

для i **от** 1 **до** $(n-1)$ **выполнять**

для j **от** $(i+1)$ **до** n **выполнять**

$\ldots$

$\vec{F}_i \leftarrow \vec{F}_i + \ldots;$

// u_1 v_1

$\vec{F}_j \leftarrow \vec{F}_j - \ldots;$

// u_2 v_2

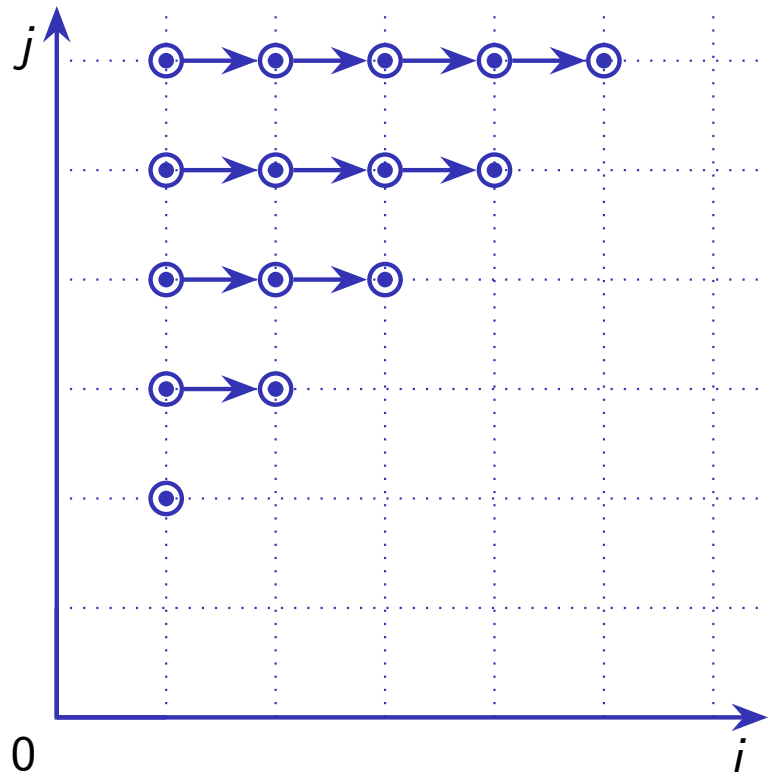

Рис. 2: Элементарный решётчатый граф вхождений (u_2, v_2)

Элементарный решётчатый граф для цикла

Пример

для i **от** 1 **до** $(n-1)$ **выполнять**
  для j **от** $(i+1)$ **до** n **выполнять**
    $\ldots$
    $\vec{F}_i \leftarrow \vec{F}_i + \ldots;$
    // u_1 v_1
    $\vec{F}_j \leftarrow \vec{F}_j - \ldots;$
    // u_2 v_2

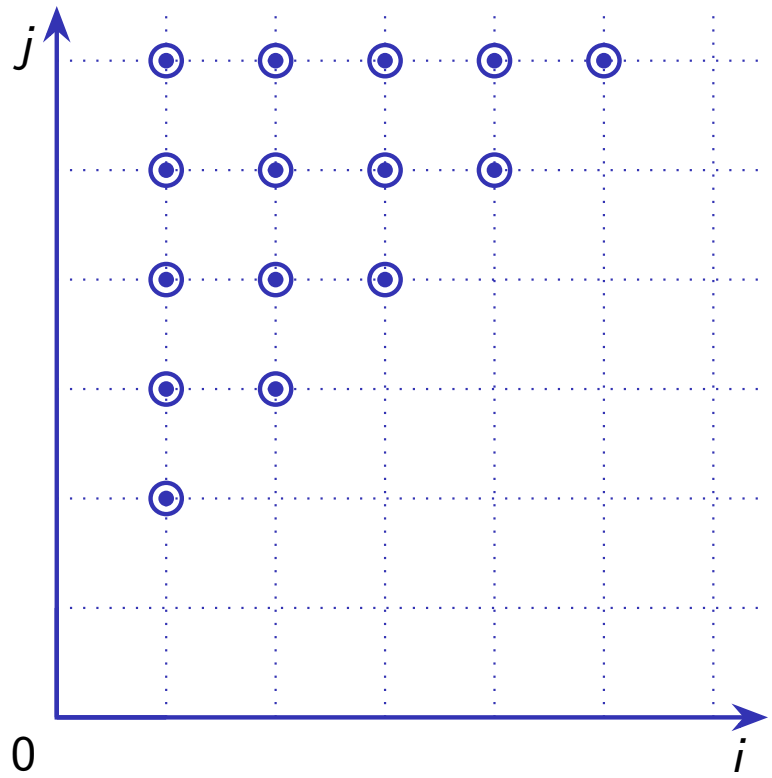

Рис. 2: Элементарный решётчатый граф вхождений (u_1, v_2)

Элементарный решётчатый граф для цикла

Пример

для i **от** 1 **до** $(n-1)$ **выполнять**
  для j **от** $(i+1)$ **до** n **выполнять**
    $\ldots$
    $\vec{F}_i \leftarrow \vec{F}_i + \ldots;$
  // u_1 v_1
    $\vec{F}_j \leftarrow \vec{F}_j - \ldots;$
  // u_2 v_2

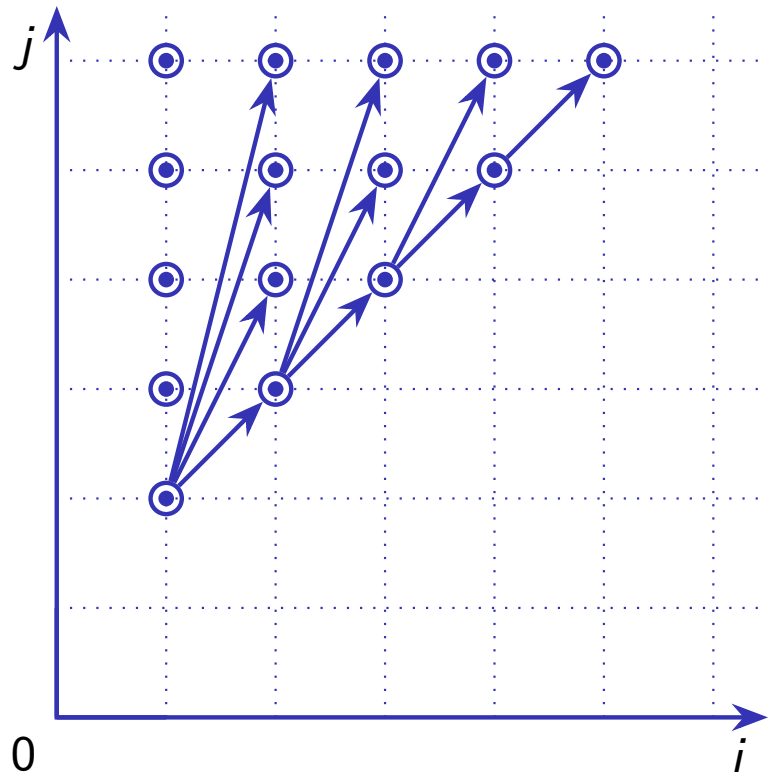

Рис. 2: Элементарный решётчатый граф вхождений (u_2, v_1)

Решётчатый граф истинной зависимости переменной

Определение

Решётчатым графом тесного гнезда циклов истинной зависимости переменной x называется граф, для которого:

1. Множеством вершин является пространство итераций тесного гнезда (D).
2. Дуга соединяет вершины i и j, если на итерации i осуществляется запись значения в некоторую ячейку $x_{f(i)}$, а на итерации j это значение из ячейки извлекается использованием.

Замечание

Аналогично определяются решётчатые графы для других видов зависимостей.

Алгоритм построения решётчатого графа

Алгоритм

1. В качестве вершин взять пространство итераций D.
2. В качестве дуг взять объединение множеств дуг элементарных решётчатых графов для всех пар вхождений переменной x в данный фрагмент программы.
3. Для каждой вершины из всех входящих в неё дуг, соответствующих зависимости по одной и той же ячейке $x_{f(i)}$, оставить только одну, начальная вершина которой лексикографически ближе всего к данной.

Пример решётчатого графа истинной зависимости

Пример

для i **от** 1 **до** $(n-1)$ **выполнять**
  для j **от** $(i+1)$ **до** n **выполнять**
    $\ldots$
    $\vec{F}_i \leftarrow \vec{F}_i + \ldots;$
  // u_1 v_1
    $\vec{F}_j \leftarrow \vec{F}_j - \ldots;$
  // u_2 v_2

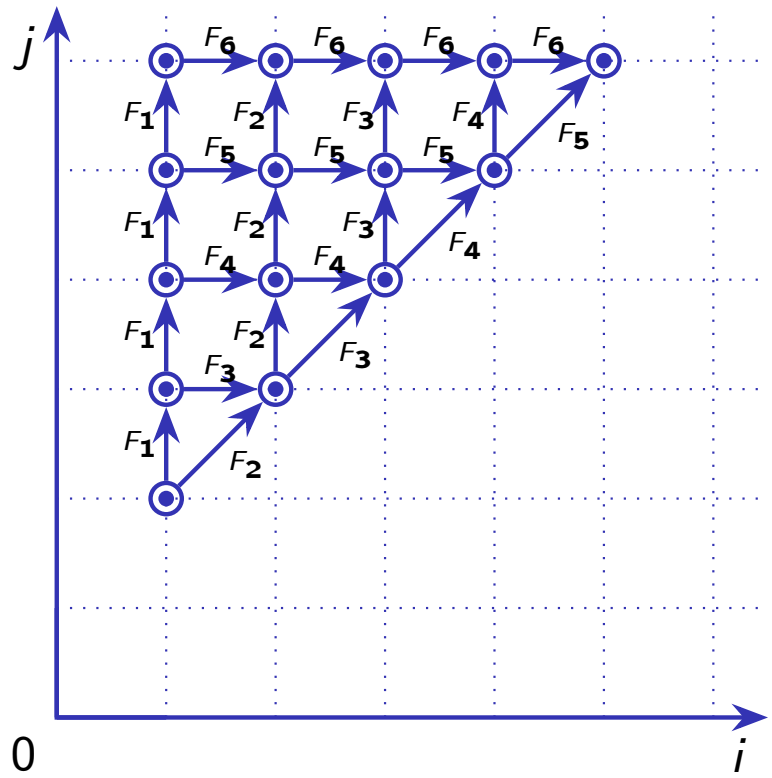

Рис. 3: Решётчатый граф истинной зависимости переменной F

Полный решётчатый граф фрагмента программы

Определение

Полным решётчатым графом фрагмента программы (заданного типа зависимости или всех типов зависимости) называется граф, для которого:

1. Множеством вершин является объединение пространств итераций всех гнёзд циклов, составляющих фрагмент (линейные участки можно считать циклами с одной итерацией).
2. Множеством дуг является объединение множеств дуг решётчатых графов всех переменных данного фрагмента (только заданного типа зависимости или всех типов).

Применение решётчатых графов

Утверждение

- *Дуги полного решётчатого графа истинных зависимостей заданного фрагмента программы указывают на необходимость пересылки данных между различными итерациями.*
- *Дуги полного решётчатого графа антизависимостей указывают на необходимость в барьерной синхронизации между различными итерациями фрагмента.*

Необходимость в барьерной синхронизации

Пример (численное дифференцирование)

$$y'_0 = \frac{y_1 - y_0}{h}, \quad y'_N = \frac{y_N - y_{N-1}}{h}, \quad y'_i = \frac{y_{i+1} - y_{i-1}}{2\,h} \quad \forall i \in [1, N-1]_{\mathbb{N}}$$

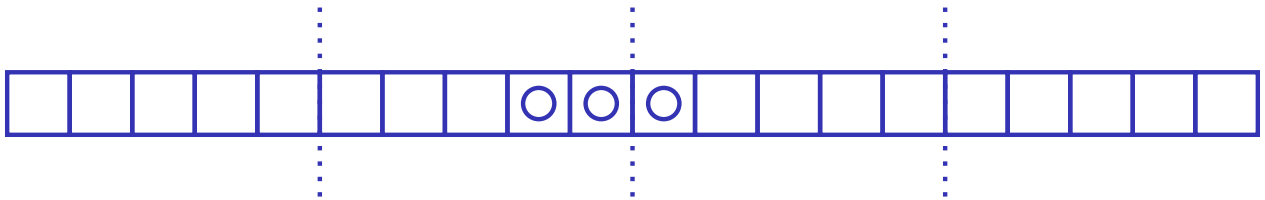

Рис. 4: Разбиение данных по вычислительным узлам

Пример распределения итераций между процессами

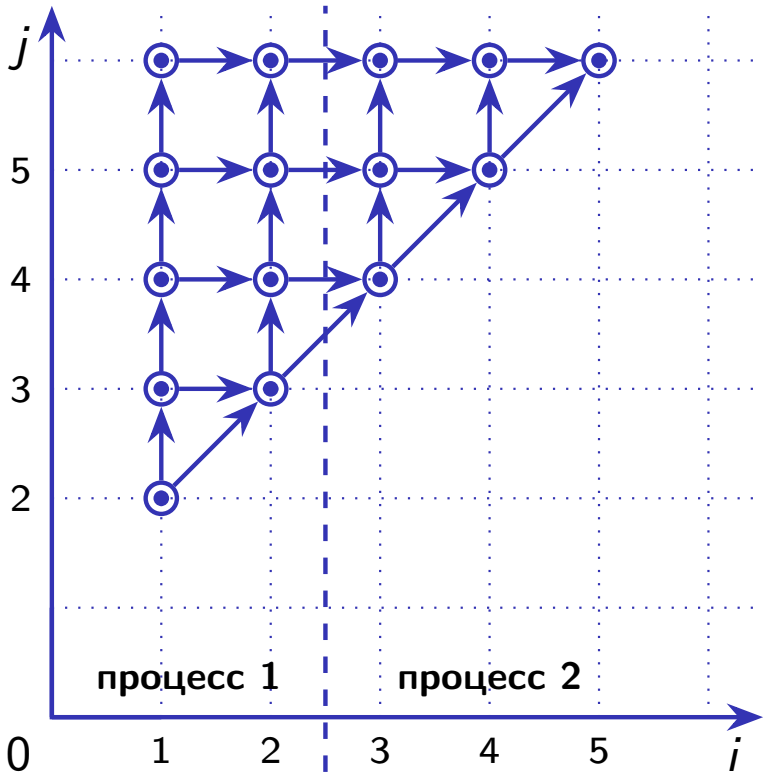

Рис. 5: Решётчатый граф истинной зависимости переменной F

Слои и параллельные множества

Определения

Слой: множество вершин графа, в котором никакие вершины не связаны путём.

Параллельные множества: множества вершин графа, для которых никакие две вершины из разных множеств не связаны путём.

Утверждение

- *Все итерации слоя можно исполнять в любом порядке (в т. ч. параллельно).*
- *Каждое множество итераций из набора параллельных множеств можно исполнять независимо от других множеств.*

Слои и параллельные множества

Определения

Слой: множество вершин графа, в котором никакие вершины не связаны путём.

Параллельные множества: множества вершин графа, для которых никакие две вершины из разных множеств не связаны путём.

Утверждение

- *Все итерации слоя можно исполнять в любом порядке (в т. ч. параллельно).*
- *Каждое множество итераций из набора параллельных множеств можно исполнять независимо от других множеств.*

Пример слоёв решётчатого графа

Пример

```
for (i = 1; i < N; ++ i)
  for (j = 1; j < N; ++ j)
    a[i][j] = f(a[i - 1][j - 1]);
```

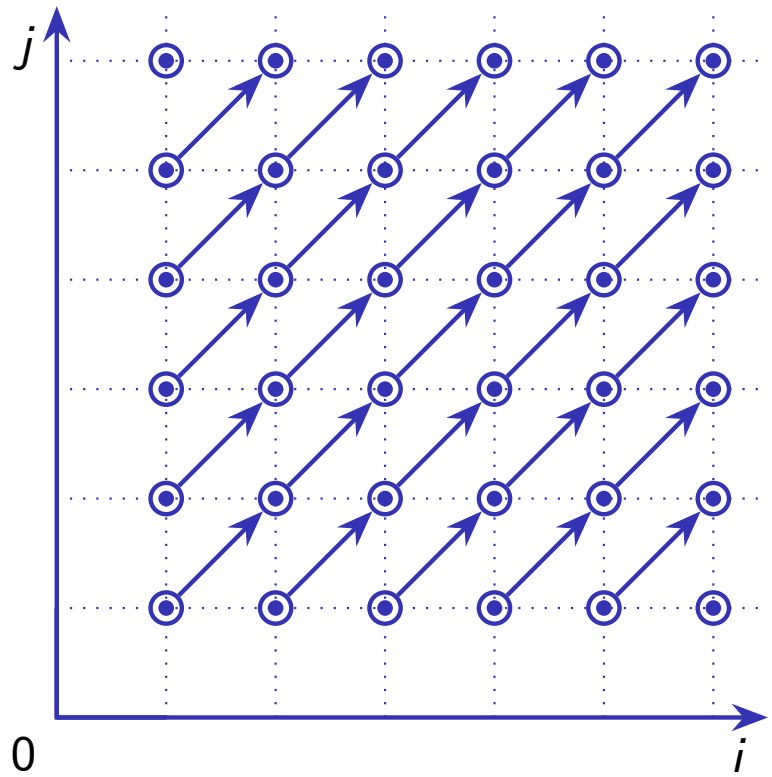

Рис. 6: Решётчатый граф фрагмента программы

Использование Intel C++ Compiler

Общий синтаксис

`icl` [⟨*настройки*⟩] ⟨*файл*$_1$⟩ [⟨*файл*$_2$⟩ ...] [`/link` ⟨*настройки_компоновки*⟩]

Ключи распараллеливания

- `/Od`, `/O1`, `/O2`, `/O3`.
- `/Qparallel` (+ `/O2` или `/O3`)
- `/Qopt-report`[:⟨номер⟩].
- `/Qopt-report-phase`[:⟨имя⟩].

Пример

```
icl /O3 /Qopt-report:5 /Qopt-report-phase:all test.c
```

Векторизация кода с возможной зависимостью

Пример (test_parallel_02.c)

```
void sum(int n, int *a, int *b, int *c)
{
  int i;
  for (i = 0; i < n; ++ i)
    a[i] = b[i] + c[i];
}
```

Пример (команда компиляции)

```
icl /O3 /S /FAs /Qopt-report:5 test_parallel_02.c
```

Отчёт о векторизации

Пример (test_parallel_02.optrpt)

```
LOOP BEGIN at ...test_parallel_01.c(6,3)
<Peeled loop for vectorization>
LOOP END
```

«Снятие кожуры» с циклов (loop peeling)

Исходный цикл

```
signed char *a =
  /* ... */;
// ...
for (i = 0; i < N; ++ i)
  loop_body(i, a[i]);
```

Очищенный цикл

```
const size_t cuAlign = 1 << 4;
size_t uPeel = (size_t) a & (cuAlign - 1);
if (uPeel != 0)
{
  uPeel = cuAlign - uPeel;
  for (i = 0; i < uPeel; ++ i)
    loop_body(i, a[i]);
}

for (i = uPeel; i < N; ++ i)
  loop_body(i, a[i]);
```

Отчёт о векторизации (продолжение)

Пример (test_parallel_02.optrpt, продолжение)

```
LOOP BEGIN at ...test_parallel_02.c(6,3)
<Multiversioned v1>
   remark: Loop multiversioned for Data Dependence
   remark: vectorization support: reference a has aligned access
   remark: vectorization support: reference b has unaligned access
   remark: vectorization support: reference c has aligned access
   remark: vectorization support: unaligned access used inside loop body
   remark: vectorization support: vector length 4
   ...
   remark: LOOP WAS VECTORIZED
   remark: entire loop may be executed in remainder
   ...
LOOP END
```

Отчёт о векторизации (продолжение)

Пример (test_parallel_02.optrpt, продолжение)

```
LOOP BEGIN at ...test_parallel_02.c(6,3)
<Alternate Alignment Vectorized Loop, Multiversioned v1>
LOOP END

LOOP BEGIN at ...test_parallel_02.c(6,3)
<Remainder loop for vectorization, Multiversioned v1>
LOOP END
```

Отчёт о векторизации (окончание)

Пример (test_parallel_02.optrpt, окончание)

```
LOOP BEGIN at ...test_parallel_02.c(6,3)
<Multiversioned v2>
   remark: loop was not vectorized: non-vectorizable loop instance from
   multiversioning
   remark: unrolled with remainder by 2
LOOP END

LOOP BEGIN at ...test_parallel_02.c(6,3)
<Remainder, Multiversioned v2>
LOOP END
```

Векторизация кода без зависимостей

Пример (test_parallel_03.c)

```
void sum(int n, int *restrict a, int *restrict b, int *restrict c)
{
  int i;
// #pragma ivdep
  for (i = 0; i < n; ++ i)
    a[i] = b[i] + c[i];
}
```

Пример (команда компиляции)

```
icl /Qstd=c11 /O3 /S /FAs /Qopt-report:5 test_parallel_03.c
```

Векторизация кода с выровненными данными

Пример (test_parallel_04.c)

```
void sum(int n, int *restrict a, int *restrict b, int *restrict c)
{
  int i;
#pragma vector aligned
  for (i = 0; i < n; ++ i)
    a[i] = b[i] + c[i];
}
```

Пример (команда компиляции)

```
icl /Qrestrict /O3 /Qunroll /S /FAs /Qopt-report:5 test_parallel_04.c
```

Векторизация с выровненными данными (окончание)

Пример (test_parallel_06.c)

```
void sum(int n, int *restrict a, int *restrict b, int *restrict c)
{
  int i;
  //
#ifdef __INTEL_COMPILER
  __assume_aligned (a, 16);
  __assume_aligned (b, 16);
  __assume_aligned (c, 16);
#endif
  //
  for (i = 0; i < n; ++ i)
    a[i] = b[i] + c[i];
}
```

Объявление выровненных данных

Пример (test_parallel_04_main.c)

```
#include <stdio.h>

void sum(int n, int *restrict a, int *restrict b, int *restrict c);

int main()
{
#ifdef _WIN32
  typedef _declspec (align (16)) int array[100];
#else
  typedef int array[100] __attribute__ ((aligned (16)));
#endif
  //
```

Объявление выровненных данных (окончание)

Пример (test_parallel_04_main.c, окончание)

```
  int i;
  array a, b, c;
  for (i = 0; i < 100; ++ i)
    b[i] = c[99 - i] = i;
  //
  sum(100, a, b, c);
  //
  for (i = 0; i < 100; ++ i)
    printf("% 3d ", a[i]);
  printf("\n");
}    // main()
```

Невекторизуемый код

Пример (test_parallel_05.c)

```
void sum(int n, int *restrict a, int *restrict b, int *restrict c)
{
  int i;
#pragma vector aligned
  for (i = 0; i < n && b[i] != 0; ++ i)
    a[i] = b[i] + c[i];
}
```

Отчёт о компиляции невекторизуемого кода

Пример (test_parallel_05.optrpt)

```
LOOP BEGIN at ...test_parallel_05.c(5,3)
   remark: loop was not vectorized: loop with multiple exits cannot be
   vectorized unless it meets search loop idiom criteria
LOOP END
```

Рекомендации по написанию векторизуемых циклов

Требования к циклам

- Количество итераций известно на начало цикла ($\Rightarrow$ 1 вход, 1 выход).
- Из ветвлений допустимы только маскируемые условные операторы.
- Самый вложенный цикл гнезда.
- Из функций только **inline** и встроенные (<cmath>).

Пример

```
void sum(
  int n, int *restrict a,
  int *restrict b, int *restrict c)
{
  int i;
#pragma vector aligned
  for (i = 0; i < n; ++ i)
  {
    a[i] = b[i] + c[i];
    if (a[i] < 0)
      break;
  }
}
```

Рекомендации по написанию векторизуемых циклов

Требования к циклам

- Количество итераций известно на начало цикла ($\Rightarrow$ 1 вход, 1 выход).
- Из ветвлений допустимы только маскируемые условные операторы.
- Самый вложенный цикл гнезда.
- Из функций только **inline** и встроенные (<cmath>).

Пример

```
void sum(
  int n, int *restrict a,
  int *restrict b, int *restrict c)
{
  int i;
#pragma vector aligned
  for (i = 0; i < n; ++ i)
  {
    int t = b[i] + c[i];
    if (t > 0)
      a[i] = t;
  }
}
```

Рекомендации по написанию векторизуемых циклов

Требования к циклам

- Количество итераций известно на начало цикла ($\Rightarrow$ 1 вход, 1 выход).
- Из ветвлений допустимы только маскируемые условные операторы.
- Самый вложенный цикл гнезда.
- Из функций только `inline` и встроенные (<cmath>).

Рекомендации по написанию векторизуемых циклов

Требования к циклам

- Количество итераций известно на начало цикла ($\Rightarrow$ 1 вход, 1 выход).
- Из ветвлений допустимы только маскируемые условные операторы.
- Самый вложенный цикл гнезда.
- Из функций только **`inline`** и встроенные (`<cmath>`).

Влияние информационных зависимостей

Зависимости

Вид	Разрешена
истинная	редукция
анти-	в пределах 1 инструкции
входная	✓
выходная	редукция

Пример

```
a[0] = 0;
for (i = 1; i < n; ++ i)
  a[i] = a[i - 1] + 1;

sum = 0;
for (i = 0; i < n; ++ i)
  sum = sum + b[i] * c[i];
```

Влияние информационных зависимостей

Зависимости

Вид	Разрешена
истинная	редукция
анти-	в пределах 1 инструкции
входная	✓
выходная	редукция

Пример

```
for (i = 1; i < n; ++ i)
  a[i - 1] = a[i] + 1;

for (i = 1; i < n; ++ i)
{
  a[i - 1] = a[i] + 1;
  b[i]     = a[i] * 2;
}
```

Влияние информационных зависимостей

Зависимости

Вид	Разрешена
истинная	редукция
анти-	в пределах 1 инструкции
входная	✓
выходная	редукция

Пример

```
for (i = 0; i < n; ++ i)
{
  b[i] = a[i] + 1;
  c[i] = a[i] * 3;
}
```

Влияние информационных зависимостей

Зависимости

Вид	Разрешена
истинная	редукция
анти-	в пределах 1 инструкции
входная	✓
выходная	редукция

Пример

```
for (i = 1; i < n; ++ i)
  x = a[i] + 1;

sum = 0;
for (i = 0; i < n; ++ i)
  sum = sum + b[i] * c[i];
```

Доступ к памяти с шагом

Пример

```
// "Vectorization possible but seems inefficient"
for (i = 0; i < n; i += 2)
  b[i] += a[i] * x[i];

for (j = 0; j < n; ++ j)
  for (i = 0; i < n; ++ i)
    b[i] += a[i][j] * x[j];

// "Existence of vector dependence"
for (i = 0; i < n; ++ i)
  b[i] += a[i] * x[index[i]];
```

Массивы структур и структуры массивов

Пример (AoS)

```
struct Point
{
  float m_fR;
  float m_fG;
  float m_fB;
};
```

Пример (SoA)

```
struct Points
{
  float *m_pfR;
  float *m_pfG;
  float *m_pfB;
};
```

Предпочтения для ускорения векторного кода

Предпочтения для внутренних циклов

- Инвариантное количество итераций.
- Только маскируемые условия в теле.
- Без зависимостей между итерациями (по крайней мере, истинных).
- Только векторизуемые операции и типы.

Предпочтения для ускорения векторного кода

Предпочтения для структур данных

- Выровненные данные.
- SoA.
- Циклы с единичным шагом.
- Прямое использование массивов и индексов в выражениях доступа к памяти.
- Прямое использование счётчиков циклов в индексных выражениях вместо увеличиваемых переменных.
- Без косвенной адресации.
- Счётчик внутреннего цикла используется в качестве последнего индекса массива (размещение по строкам).
- Минимальный тип данных для заданной точности.
- Без преобразований типов.

Прагмы настроек опимизаций

Прагмы векторизации

```
#pragma ivdep
#pragma vector aligned
#pragma vector unaligned
#pragma vector always
#pragma novector
#pragma loop count (1000)
```

Прагмы распараллеливания

```
#pragma distribute point
#pragma unroll
#pragma unroll (4)
#pragma nounroll
#pragma parallel
#pragma noparallel
```