



# *Анализ и оптимизация на софтуерни приложения*

Александър Пенев

Васил Василев

## Векторизация

# Съдържание

1. Какво е векторизация?
2. Примери
3. Векторизация на цикли
4. Структури или масиви
5. Векторизация на при различни типове



# Какво е векторизация?

- ❖ Векторизация (vectorization) е процес на преобразуване на програма изпълняваща една операция за всеки такт от време (в една нишка) към програма изпълняваща няколко операции едновременно в един такт (в една нишка)
- ❖ Векторизацията е частен случай на паралелизацията
- ❖ Обикновено се използват SIMD инструкциите на съвременните процесори (MMX, SSE, AVX, AltiVec, NEON, ...)



# Как се прилага

- ❖ Ръчно
- ❖ Ръчно с използване на intrinsic функции
- ❖ Автоматично от компилатора  
*Обикновено компилатора трябва да бъде „подпомогнат“ от нас*
- ❖ Автоматично от процесора  
*Като част от скаларните и супер-скаларните архитектури*



# Пример 1

```
for (i = 0; i < 1024; i++)  
    C[i] = A[i] + B[i];
```

Векторизация на цикли

```
for (i = 0; i < 1024; i+=4)  
    (C[i], C[i+1], C[i+2], C[i+3]) =  
        (A[i], A[i+1], A[i+2], A[i+3]) +  
        (B[i], B[i+1], B[i+2], B[i+3]));
```

ПСЕВДОКОД

Частично развиване на цикъл.  
Може ли границите на цикъла да не са кратни на 4?



## Пример 2

```
for (i = 0; i < MAX; i++)  
{  
    C[i].x = A[i].x + B[i].x;  
    C[i].y = A[i].y + B[i].y;  
    C[i].z = A[i].z + B[i].z;  
}
```

Векторизация в блок

```
for (i = 0; i < MAX; i++)  
{  
    (C[i].x, C[i].y, C[i].z) =  
        (A[i].x, A[i].y, A[i].z) +  
        (B[i].x, B[i].y, B[i].z);  
}
```

псевдокод

Ако дължината на вектора е 4, то в примера има проблем с непълното използване на SIMD възможностите



# Пример 3

Векторизация на  
разклонени алгоритми

```
for (i = 0; i < 1024; i++)
{
    if (A[i] > 0)
        C[i] = B[i];
    else
        D[i] = D[i-1];
}
```

```
for (i = 0; i < 1024; i++)
{
    P = A[i] > 0;
    NP = !P;
    C[i] = B[i];    (P)    // изпълнява се само ако P е истина
    D[i] = D[i-1]; (NP)    // изпълнява се само ако NP е истина
}
```

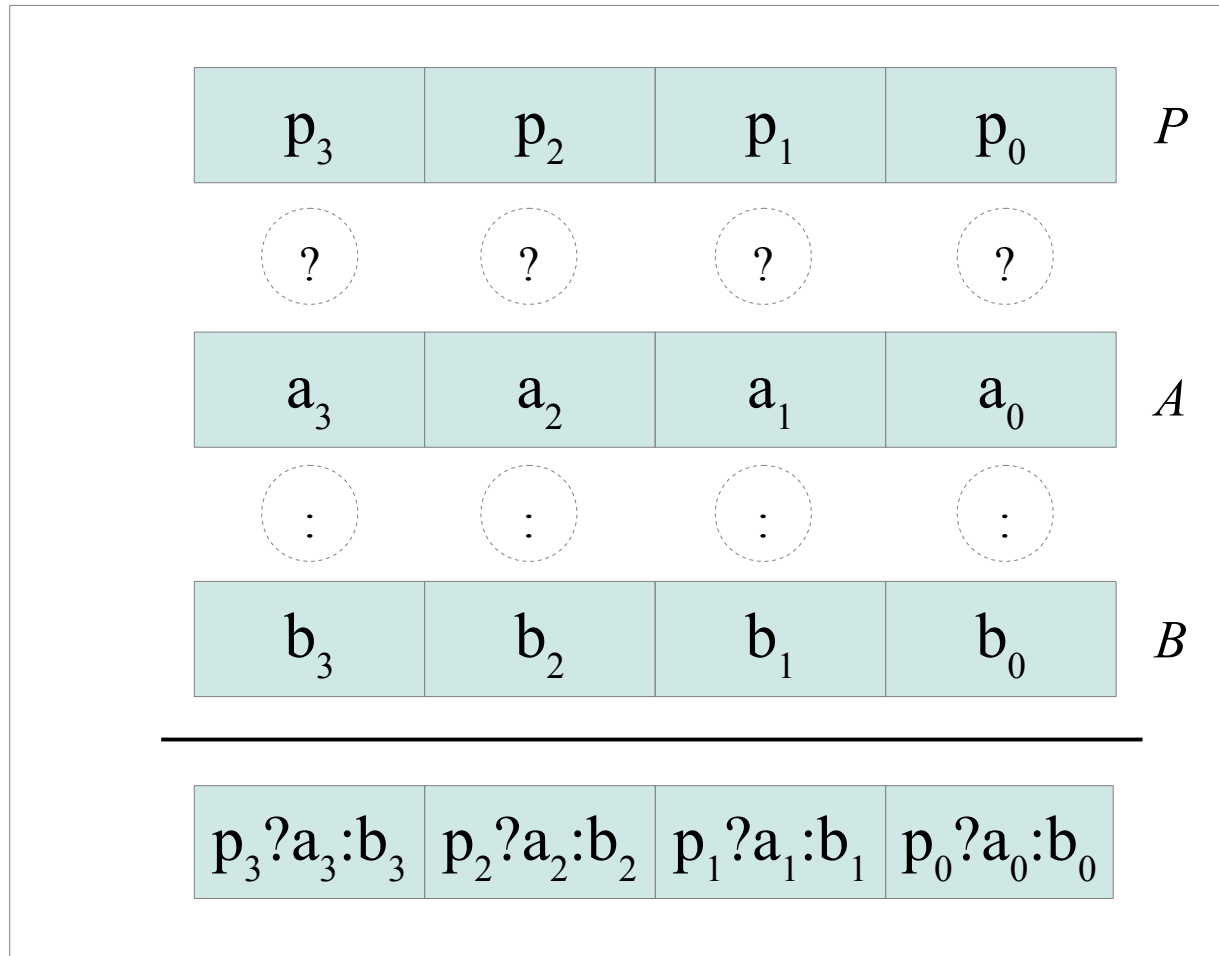
псевдокод

```
for (i = 0; i < 1024; i+=4) {
    vP = (A[i], A[i+1], A[i+2], A[i+3]) > (0, 0, 0, 0);
    vNP = !vP; // векторно т.е. покомпонентно отрицание
    (C[i], C[i+1], C[i+2], C[i+3]) =
        vsel(vP, (B[i], B[i+1], B[i+2], B[i+3]), (C[i], C[i+1], C[i+2], C[i+3]));
    if (vNP[4]) D[i+3] = D[i+2]; if (vNP[3]) D[i+2] = D[i+1];
    if (vNP[2]) D[i+1] = D[i];    if (vNP[1]) D[i] = D[i-1];
}
```

псевдокод



# Пример 3 – Векторна $\phi$ -я $vsel(P, A, B)$





# Векторизация на цикли – общ случай

Всеки цикъл се развива до определено ниво (зависи от границите т.е. дали са известни по време на компилация и колко са големи).

Циклите се разделят на 4 части (или по-малко):

## ❖ Пред цикъл

*Операции независими от цикъла (инварианти). Обикновено се зареждат (инвариантни) данни във векторни регистри и други*

## ❖ Цикъл (Цикли)

*Векторизирани вариант(и) на цикъла (циклите)*

## ❖ След цикъл

*Получаване на резултати и допълнителна инвариантна обработка. Индукции, редукции и други*

## ❖ Опашка на цикъл

*Реализация на неекторизиран вариант на цикъла за оставащите итерации, които по някакви причини не са попаднали в основния цикъл (например броя итерации не е кратен на големината на векторите или размера се определя в runtime)*



# Не всичко може да се векторизира

```
for (i = 0; i < 3; i++)  
{  
    C[i] = A[i] + B[i];  
}
```

**ОК.**

Цикълът може да бъде  
развит и/или векторизиран  
напълно

```
for (i = 0; i < 1024; i++)  
{  
    C[i] = A[i] + B[i];  
}
```

**ОК.**

Цикълът може да бъде  
векторизиран

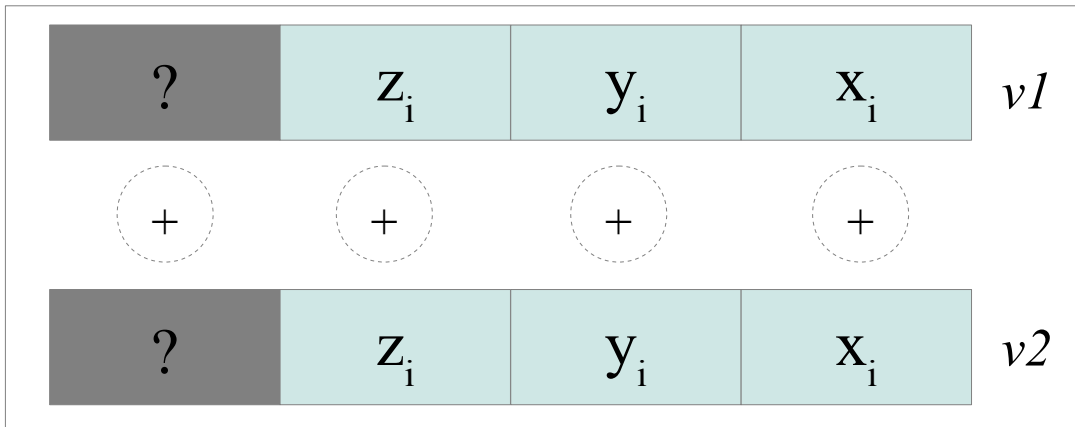
```
for (i = 0; i < 1024; i++)  
{  
    D[i] = E[i] - A[i-1];  
    A[i] = B[i] + C[i];  
}
```

**Не може!**

Използва се стойност преди  
да бъде пресметната

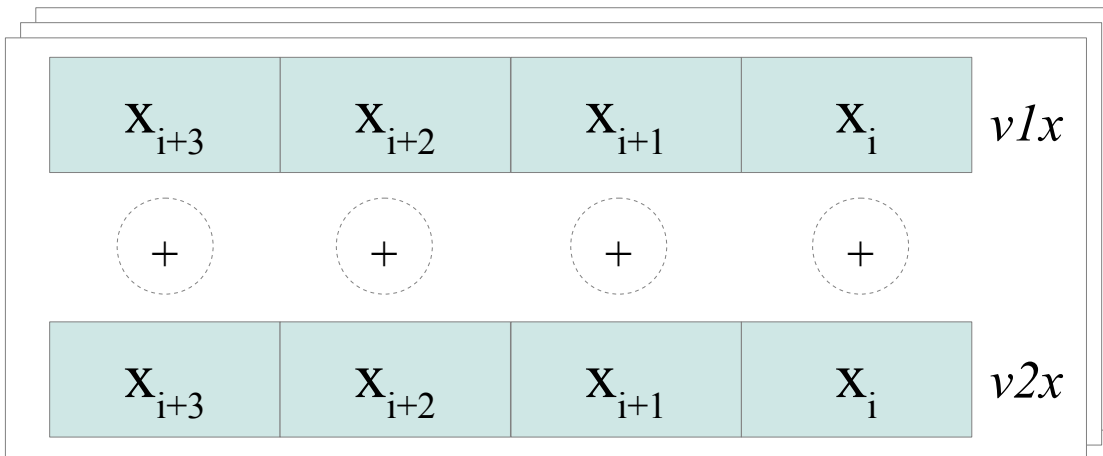


# Масив от структури или няколко масива?



```
struct { float x, y, z } vec;  
  
vec[100] v1, v2;  
  
for (i = 0; i < 100; i++) {  
    v1[i].x += v2[i].x;  
    v1[i].y += v2[i].y;  
    v1[i].z += v2[i].z;  
}
```

ИЛИ



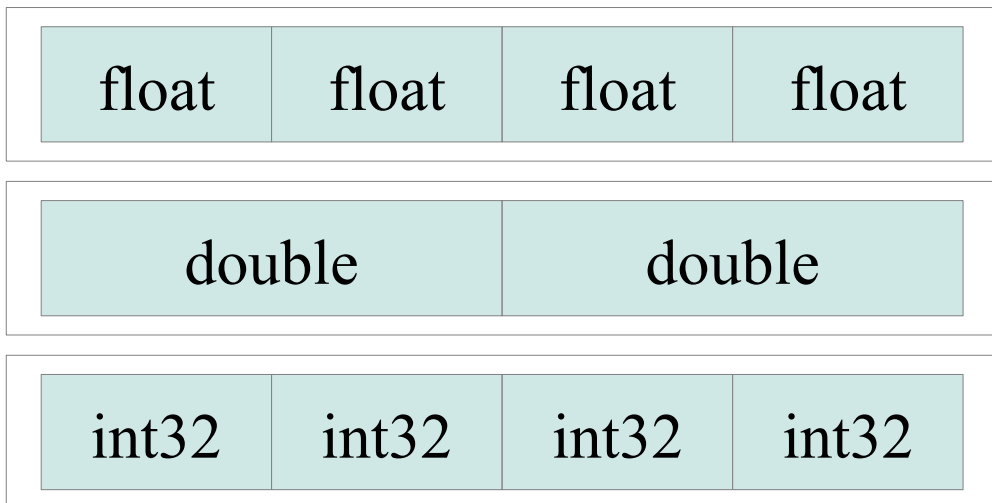
```
float[100] v1x, v2x;  
float[100] v1y, v2y;  
float[100] v1z, v2z;  
  
for (i = 0; i < 100; i++)  
    v1x[i] += v2x[i];  
for (i = 0; i < 100; i++)  
    v1y[i] += v2y[i];  
for (i = 0; i < 100; i++)  
    v1z[i] += v2z[i];
```



# Типове данни

## ❖ Ако типа е еднакъв

Например 4 float числа се събират точно в един SSE регистър



## ❖ Ако типовете са различни

Имаме загуба на производителност. Трябва да се внимава с Align на структурите  
За някои типове може да се налага разширяване до по-голям тип (float->double)

