



**Нижегородский государственный университет
им. Н.И.Лобачевского**

Факультет Вычислительной математики и кибернетики

***Параллелизм
как основа архитектуры ВС***

Раздел 9

Лабораторная работа

SIMD: перспективы производительности

Кудин А.В., к.т.н.

Содержание

- ❑ Описание задачи
- ❑ Исходное решение задачи
- ❑ Постановка задачи: ускорение существующего решения
- ❑ Векторное решение задачи
- ❑ Измерение ускорения



Описание задачи

У органов специального назначения имеются образцы речи некоторых граждан. Необходимо автоматически распознать принадлежность произнесённого слова тому или иному гражданину.

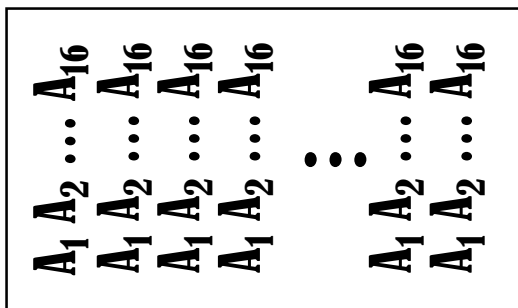
Для этого образцы речи переводятся из пространственной области в область параметров фильтра предсказания некоторого вокодера (voice coder). Каждый отсчёт в области параметров – вектор из 16 однобайтовых элементов.

Идентификация осуществляется путём свёртки слова на каждом фрагменте каждого образца и сравнения найденного минимума в ортогональной метрике с пороговым значением. При большой базе данных скорость работы идентификатора является превалирующей.



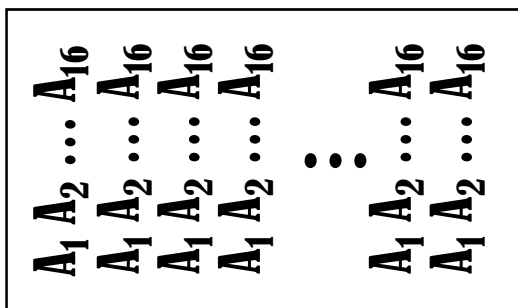
Описание задачи в картинках

Образец 1:

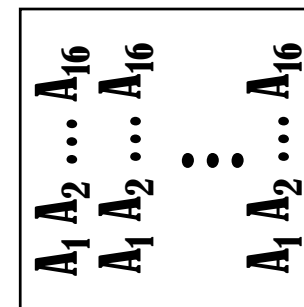


...

Образец N:



Сигнатура слова:

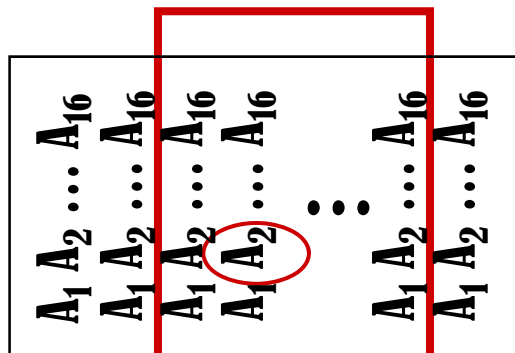


ПОИСК

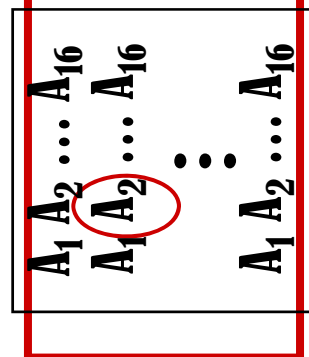
Свёртка слова (функция Stage)

Свёртка сигнатуры (слова) на фрагменте поисковых признаков (образца) в ортогональной метрике

Образец К:



Сигнатура слова:



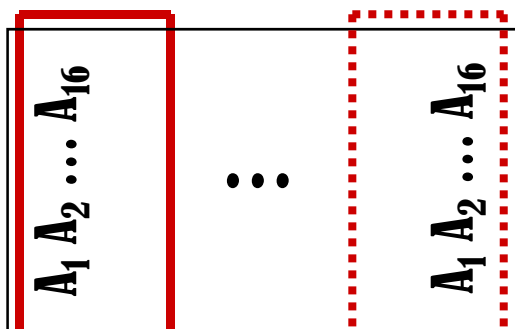
Stage: вычисление суммы модулей разностей всех пар элементов для текущего фрагмента образца



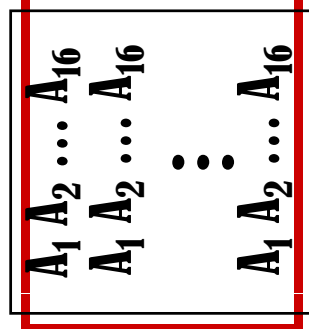
Вычисление расстояния (функция SignFind)

Движение по поисковым признакам (по образцу) и
нахождение минимума (определение расстояния)

Образец К:



Сигнатура слова:



SignFind: вычисление свёртки
на каждом фрагменте текущего
образца, определение
минимума и сравнение с
пороговым значением



Спецификация функции Stage

```
int Stage (void * original, void * signature, int signature_leng);
```

```
// свёртка сигнатуры на фрагменте поисковых признаков
```

```
// original – указатель на фрагмент поисковых признаков (кратно 16)
```

```
// signature – указатель на сигнатуру (кратно 16)
```

```
// signature_leng – число 128-битных векторов в сигнатуре
```

```
// возвращаемое значение – величина расстояния
```



Реализация функции Stage на Си

```
int Stage (void * original, void * signature, int signature_leng)
{
    int i, j;
    int S=0;
    for (j=0; j<signature_leng; j++)
        for (i=0; i<16; i++)
            S += abs( (int)((unsigned char *) original + 16 * j + i )) –
                    (int)((unsigned char *) signature + 16 * j + i )) );
    return S;
}
```



Спецификация функции SignFind

```
int SignFind (void * original, int original_leng, void * signature, int signature_leng);
```

```
// поиск в признаках
```

```
// original – указатель на поисковые признаки (кратно 16)
```

```
// original_leng – число 128-битных векторов
```

```
// signature – указатель на сигнатуру (кратно 16)
```

```
// signature_leng – число 128-битных векторов
```

```
// возвращаемое значение – минимальная величина расстояния
```

```
// (или -1, если правдоподобие не найдено)
```



Реализация функции SignFind на Си

```
int SignFind (void * original, int original_leng, void * signature, int signature_leng)
{
    char * U;
    char * V = (char *)original + ((original_leng-signature_leng)<<4);
    int Smin = THRESHOLD;

    for (U=original; U<=V; U+=16)
    {
        int S = Stage(U, signature, signature_leng);
        if (S<Smin) Smin=S;
    }
    return Smin < THRESHOLD ? Smin : -1 ;
}
```



Постановка задачи

Производительность приведённой функции SignFind неприемлема.

Необходимо:

1. Определить критичный по производительности участок кода
2. Определить возможность применения векторной обработки
3. Разработать SIMD алгоритм
4. Реализовать SIMD алгоритм для Pentium 4
5. Сравнить производительность по схеме K-best



Решение задачи

критичный по производительности участок кода – базовый блок инструкций (ББИ) – наиболее глубоко вложенный код, вызываемый наиболее часто – тело функции Stage

данный ББИ хорошо сопрягается с технологией SSE



Реализация функции Stage на ассемблере Pentium 4 с использованием техники SSE в синтаксисе AT&T

```
.text
.global Stage
Stage:
```

```
    enter    $0, $0
    push     %esi
    push     %edi

    mov      8(%ebp), %edi      ; edi – указатель на фрагмент поисковых признаков
    mov      12(%ebp), %esi     ; esi – указатель на сигнатуру
    mov      16(%ebp), %ecx     ; ecx – число 128-битных векторов в сигнатуре

    PXOR     %xmm1, %xmm1      ; xmm1 – аккумулятор суммы разностей

1:    MOVDQA  (%edi), %xmm0      ; загрузка вектора признаков
    add      $16, %edi
    PSADBW   (%esi), %xmm0      ; загрузка вектора сигнатуры и вычисление суммы модулей разностей
    add      $16, %esi
    PADDD    %xmm0, %xmm1      ; пополнение аккумулятора
    loop     1b

    PREFETCHT2 (%edi)          ; предвыборка для ускорения последующих вызовов Stage

    PSHUFD   $0xAA, %xmm1, %xmm0
    MOVD     %xmm1, %ecx        ; выгрузка первой полусуммы
    MOVD     %xmm0, %eax        ; выгрузка второй полусуммы
    add      %ecx, %eax         ; формирование результата

    pop      %edi
    pop      %esi
    leave
    ret
```



Сравнение производительности по схеме K-best

Ускорение вычислений в 51 раз!



Заключение

Дидактическая цель данной лабораторной работы:

- ❑ Практика определения критичных по производительности участков кода
- ❑ Практика применения технологии векторной обработки SSE
- ❑ Практика использования языка ассемблера
- ❑ Практика употребления мобильного синтаксиса AT&T
- ❑ Практика измерения производительности по схеме K-best

