Средства и системы параллельного программирования

Семинар #5
Основы векторизации

Как подключить векторизацию компилятором?

`gcc -Q --help=optimizers -O3 -` посмотреть, какие оптимизации включены в соответствующий уровень оптимизации (нас интересует -ftree-vectorize)

Дополнительные опции

gcc -O3 -fdump-tree-optimized (-fno-tree-loop-distribute-patterns) - в промежуточном представлении (IR) программы (уже не C/C++, ещё не ассемблер), можно увидеть в соответствующем базовом блоке, во что преобразовались соответствующие циклы

gcc -O3 -ftree-vectorize -fno-builtin-memcpy -fno-builtin-memmove - постараться, чтобы встречаемые в программе циклы были представлены в виде математических операций, а не в виде функций копирования, где это возможно

Отчёт компилятора по векторизации

-fopt-info-vec-optimized -fopt-info-vec-missed - вместе с флагами компиляции приложения вывести информацию о том, какие операции удалось векторизовать, а какие нет

Через опцию --save-temps можно посмотреть на код ассемблера (.s) и найти операции, являющиеся векторизованными

-fdump-tree-vect-details - посмотреть подробный отчёт по векторизации

Циклы

```
int i;
for ( i = 1; i < n; i++)
{
    a[i] = (i % b[i]);
}
```

```
int i;
for (i = 1; i < n - 3; i += 4)
{
    a[i] = (i % b[i]);
    a[i + 1] = ((i + 1) % b[i + 1]);
    a[i + 2] = ((i + 2) % b[i + 2]);
    a[i + 3] = ((i + 3) % b[i + 3]);
}
```

Развёртка циклов

```
int i;
for ( i = 1; i < n; i++)
{
    a[i] = (i % b[i]);
}
```

```
int i;
for (i = 1; i < n - 3; i += 4)
{
    a[i] = (i % b[i]);
    a[i + 1] = ((i + 1) % b[i + 1]);
    a[i + 2] = ((i + 2) % b[i + 2]);
    a[i + 3] = ((i + 3) % b[i + 3]);
}
```

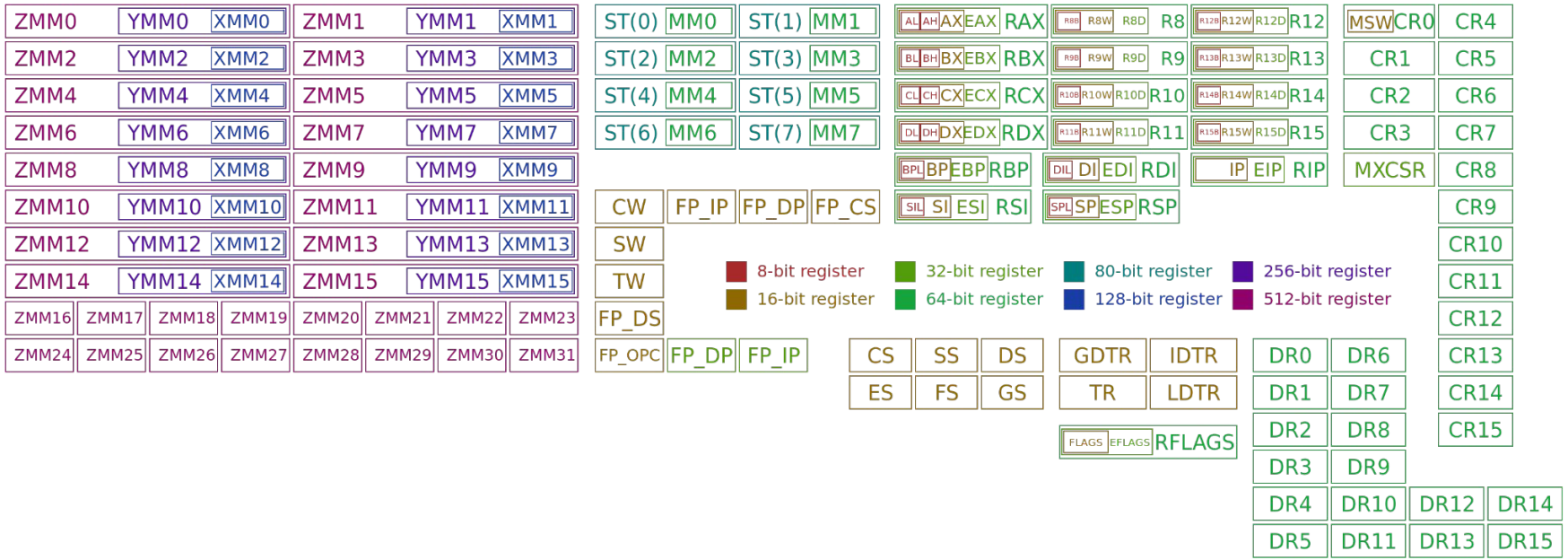
Для чего мы делаем развертку циклов:

- убираем лишние операции (на проверку границ цикла)
- убираем зависимости между последовательными операциями

Доступные на процессоре векторные инструкции

`__builtin_cpu_supports("вид инструкции")` - функция, возвращающая 1 при доступности соответствующих векторных инструкций, 0 - при недоступности

Система регистров в современных x86-процессорах



Выделение выровненного адреса

```
{
    void *mem = malloc(1024+16);
    void *ptr = ((char *)mem+16) & ~ 0x0F;
       // Use aligned pointer
    free(mem);
}
```

int posix_memalign(void **memptr, size_t alignment, size_t size); - для POSIX-совместимых систем

void *aligned_alloc(size_t alignment, size_t size); - *C11*

void* _mm_malloc (int size, int align) - *C11*

Интринсики SSE, AVX

Документация по интринсикам SSE и AVX (VPN)

https://www.laruence.com/sse/ (зеркало, можно без VPN)

На приведённых выше страницах удобно искать интринсики по действию (например, compare), которое вы хотите сделать.

x86 Intrinsics Cheat Sheet - неплохая сводка, но найти нужное может быть труднее

ARM NEON

Размер регистра SIMD = 128 байт

Документация по интринсикам ARM NEON

Задание

Реализовать операцию над вектором, производящую деление всех элементов вектора на максимальный элемент. (То есть, сначала найти максимальный элемент вектора, затем поделить все элементы на него)

Тип элемента - float или double (по желанию + смотрите на возможности вашей платформы (длину регистров)). Программа должна работать при длине вектора, не кратной числу элементов в регистре (то есть, обработать хвосты скалярно)

Задание необходимо выполнять на локальной машине с помощью интринсик используемой целевой архитектуры (AVX для Intel, AMD; NEON для ARM). В отчёте необходимо написать, какие инструкции доступны у вас.

Сравнить время выполнения программы с последовательной версией (не использующей векторизацию), представить результаты сравнения в отчёте. Для замеров рекомендуется проделать данную операцию несколько раз (ну или взять большой входной вектор).

Дедлайн: 27.10; 3.11