Оптимизация производительности вычислительных приложений

Практическое занятие №2

Векторизация

План

- Что такое векторизация
- Векторные процессоры
- Векторизация в процессорах Intel
- Автоматическая векторизация с использованием компилятора Intel
- Зависимости

Векторизация

- Векторизация – выполнение операций над несколькими операндами (вектором) одновременно
- Пример:
  - Скалярная операция: a[1] + b[1]
  - Векторная операция:

a[1] a[2] a[3] a[4]

\+ + + +

b[1] b[2] b[3] b[4]

Векторизация

- Частный случай параллельных вычислений модели SIMD
- SIMD (Single Instruction Multiple Data) – выполнение одной команды, обрабатывающей несколько данных

Векторные процессоры

- Первые суперкомпьютеры использовали векторные процессоры:
  - Специализированные процессоры для векторной обработки
  - Высокая скорость работы с памятью
  - Нет кэша
- Примеры:
  - Cray — 1 — первый суперкомпьютер компании CRAY
  - NEX SX-9 — современный векторный компьютер

Cray - 1

NEC SX-9

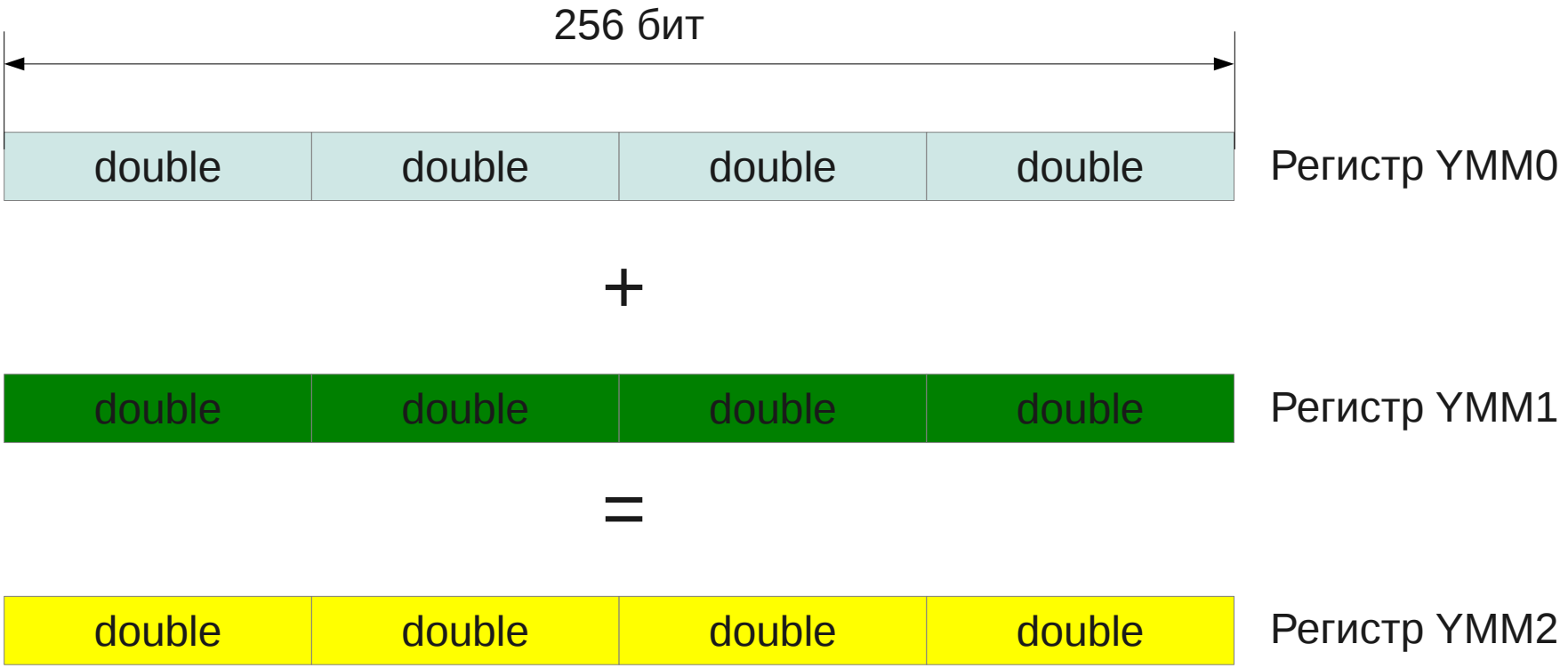
Векторизация в современных процессорах

- Современные процессоры стандартной архитектуры (Intel и AMD) имеют векторные расширения:
  - AVX
  - SSE
  - MMX
- Векторное расширение:
  - Набор векторных регистров
  - «Упаковка» — запись в один векторный регистр нескольких операндов — формирование вектора
  - Набор команд для векторной обработки

AVX

- AVX (Advanced Vector eXtensions) – современное векторное расширение в процессорах Intel и AMD
- Основные компоненты AVX:
  - Векторные регистры YMMX длиной 256 бит
  - Набор команд FMA для выполнения векторных операций
  - Новый тип кодирования инструкций
- Версии AVX:
  - AVX (Core 2 поколения), AVX2 (Core 4 поколения)

Векторные операции



Пакет и векторные операции

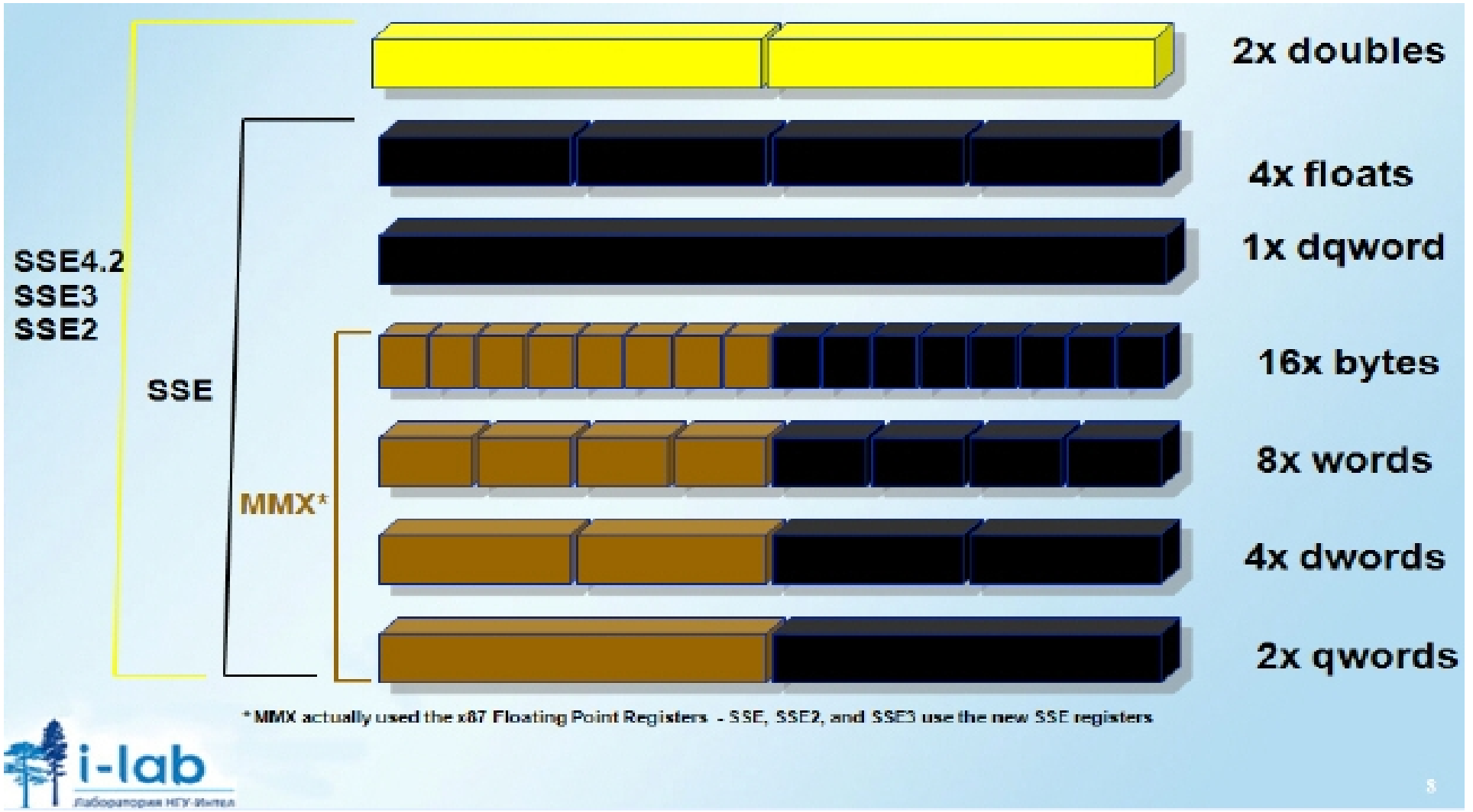
- Пакет – набор независимых операндов, записанных в векторные регистр
- От размера пакета зависит достижимое ускорение

Тип данных	Размер данных, бит	Количество данных в пакете AVX
double	64	4
float	32	8
int	16	16

Предыдущие векторные расширения

- SSE - Streaming SIMD Extensions
  - SSE4.2, SSE4.1, SSE3, SSE2, SSE
- MMX – Multimedia Extensions
  - Самое первое векторное расширение
  - Впервые появились в Pentium MMX
  - Работа только с целыми числами
- Векторные регистры длиной 128 бит

Пакеты в SSE и MMX



Проверка поддержки векторизации

```
$ grep flags /proc/cpuinfo

flags           : fpu vme de pse tsc msr pae mce
cx8 apic mtrr pge mca cmov pat pse36 clflush dts
acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx
rdtscp lm constant_tsc arch_perfmon pebs bts
rep_good xtopology nonstop_tsc aperfmperf pni
pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3
cx16 xtpr pdcm sse4_1 sse4_2 x2apic popcnt xsave
avx lahf_lm ida arat epb xsaveopt pln pts dts
tpr_shadow vnmi fl
```

Способы использования векторизации

- Ассемблер
  - Прямое использование инструкций из набора AVX
- Интринсики
  - Вставка в программу на C/C++ фрагментов, близких к ассемблеру
- Автоматическая векторизация
  - Выполняется компилятором без участия программиста
  - Компилятор не всегда находит возможность для векторизации
  - Программист может подсказать компилятору

Векторизация в компиляторе Intel

- Компилятор Intel поддерживает автоматическую векторизацию программ на C/C++ и Фортране
- Векторизация выполняется на уровнях оптимизации -O2 и -O3
- Возможна генерация векторизованного кода для разного набора инструкций:
  - AVX
  - SSE

Пример программы

```
#define N 1000000


int main()
{
    double *a, *b;
    int i;
    a = (double*)malloc(N * sizeof(double));
    b = (double*)malloc(N * sizeof(double));

    for (i = 0; i < N; i++)
   {
      sum += a[i] * b[i];
   }
   printf("Sum is %f, Duration is %f\n", sum, duration);

   free(a);
   free(b);
   return 0;
}
```

Результаты работы

- Без векторизации:

```
$ icc -O0 vect1.c -o vect1
$ ./vect1
Sum is 249824.726625, Duration is 0.003466
```

- С векторизацией:

```
$ icc -O2 vect1.c -o vect1
$ ./vect1
Sum is 249803.549353, Duration is 0.001196
```

Отчет о векторизации

- Как узнать, векторизовал ли программу компилятор, и если да, то что именно векторизовал?
- Отчет о векторизации:
  - Опция -vec-report
- Пример использования:

```
$ icc -O2 -vec-report vect1.c -o vect1

vect1.c(34): (col. 2) remark: LOOP WAS VECTORIZED.
```

Уровни отчета о векторизации

- -vec-report0 – нет отчета
- -vec-report1 – успешно векторизованные циклы
- -vec-report2 – плюс циклы, которые не удалось векторизовать
- -vec-report3 – плюс информация о зависимостях
- -vec-report4 - циклы, которые не удалось векторизовать
- -vec-report5 – плюс информация о зависимостях
- -vec-report6 – расширенная диагностическая информация

Пример отчета

```
$ icc -O2 -vec-report3 vect1.c -o vect1
vect1.c(25): (col. 2) remark: loop was not vectorized:
existence of vector dependence.

vect1.c(28): (col. 18) remark: vector dependence: assumed
OUTPUT dependence between  line 28 and  line 27.

vect1.c(27): (col. 18) remark: vector dependence: assumed
OUTPUT dependence between  line 27 and  line 28.

vect1.c(27): (col. 18) remark: vector dependence: assumed
OUTPUT dependence between  line 27 and  line 28.

vect1.c(28): (col. 18) remark: vector dependence: assumed
OUTPUT dependence between  line 28 and  line 27.

vect1.c(34): (col. 2) remark: LOOP WAS VECTORIZED.
```

Выбор типа векторного расширения

- Компилятор Intel содержит опции для выбора типа инструкций, для которого генерируется код
- Возможные типы инструкций:
  - AVX, SSE4.2, SSE4.1, SSSE3, SSE3, SSE2
- Опции:
  - -x – генерирует код только для данного типа инструкций
  - -m – генерирует код для заданного типа инструкций с проверкой поддержки их процессором
  - -ax – генерирует код для заданного типа инструкций, а также универсальную версию кода
  - -xhost – генерирует код для компьютера, на котором запускается с максимальным уровнем инструкций

Примеры:

- Код только для инструкций AVX:
  - `$ icc -O2 -xAVX vect1.c -o vect1`
- Код для инструкций AVX с альтернативной версией для процессоров не от Intel:
  - `$ icc -O2 -axAVX vect1.c -o vect1`
- Код для инструкций SSE4.2 с проверкой совместимости:
  - `$ icc -O2 -mSSE4.1 vect1.c -o vect1`

Векторизация в gcc

- gcc поддерживает автоматическую векторизацию
  - Уровень оптимизации -O3
  - Отчет о векторизации -ftree-vectorizer-verbose=x
- Пример:
  - `gcc -O3 -ftree-vectorizer-verbose=2 -ffast-math vect1.c -o vect_gcc`
  - Опция -ffast-math нужна для поддержки редукции с плавающей точкой

Практическое задание №1

- Написать программу вычисления векторного произведения векторов
- Скомпилировать программу без оптимизации и замерить время выполнения
- Скомпилировать программу с оптимизацией -O2 и включённым отчётом о векторизации
- Замерить время выполнения векторизованной программы
- Сравнить производительность векторизации для типов данных float и double
- Сгенерировать код для разного набора векторных инструкций и сравнить производительность
- (Опционально) Векторизовать программу с использованием gcc и сравнить производительность с icc

Зависимости

- Возможна ли векторизация такого цикла:

```
b[0] = a[0];
for (i = 1; i < N; i++)
{
    b[i] = b[i-1] + a[i];
}
```

Зависимости

- Отчет о векторизации:

```
    (col. 2) remark: loop was not vectorized:
existence of vector dependence.

    (col. 2) remark: vector dependence: assumed
FLOW dependence between b line 43 and b line 43.
```

- Вывод
  - Цикл не может быть векторизован, т.к. Существует зависимость по переменной b

Зависимости

- Возможна ли векторизация такого цикла:

```
for (i = 0; i < N; i++)
{
    a[i] = (double)rand() / RAND_MAX;
    b[i] = (double)rand() / RAND_MAX;
}
```

Зависимости

- Зависимость по данным ложная
  - В C указатели могут указывать на один и тот же массив
- Подсказка компилятору игнорировать возможные зависимости:
  - #pragma ivdep
  - Если зависимость есть, то программа будет работать с ошибками

Векторизованные функции

acos	ceil	fabs	round
acosh	Cos	floor	sin
asin	Cosh	fmax	sinh
asinh	erf	fmin	sqrt
atan	Erfc	log	tan
atan2	Erfinv	log10	tanh
atanh	Exp	log2	trunc
cbrt	exp2	pow	

Источник: A Guide to Vectorization with Intel® C++ Compilers

Практическое задание №2

- Модифицировать программу умножения векторов так, чтобы цикл инициализации векторов также был векторизован

Дополнительные материалы

- Приведённые примеры векторизации очень простые!
- Проблемы в реальных приложениях:
  - “Правильный” цикл
  - Не последовательный доступ к памяти
  - Зависимости
  - Выравнивание данных

Дополнительные материалы

- Using Intel® AVX without Writing AVX
  - http://software.intel.com/en-us/articles/using-intel-avx-without-writing-avx
- A Guide to Vectorization with Intel® C++ Compilers
  - http://software.intel.com/sites/default/files/m/d/4/1/d/8/CompilerAutovectorizationGuide.pdf
- Intel® Advanced Vector Extensions Programming Reference
  - http://software.intel.com/sites/default/files/m/f/7/c/36945
- Auto-vectorization in GCC
  - http://gcc.gnu.org/projects/tree-ssa/vectorization.html