Уральский федеральный университет

Параллельные вычисления
Векторизация

Созыкин Андрей Владимирович

к.т.н.
Заведующий кафедрой высокопроизводительных компьютерных технологий
Институт математики и компьютерных наук

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 компьютеры

Первые суперкомпьютеры использовали векторные процессоры:

- Специализированные процессоры для векторной обработки
- Высокая скорость работы с памятью
- Нет кэша

Примеры:

- Cray-1 первый суперкомпьютер компании CRAY
- NEC SX-ACE современный векторный компьютер

Современные процессоры

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

Векторное расширение AVX

AVX (Advanced Vector eXtensions) – современное векторное расширение в процессорах Intel и AMD

Основные компоненты AVX:

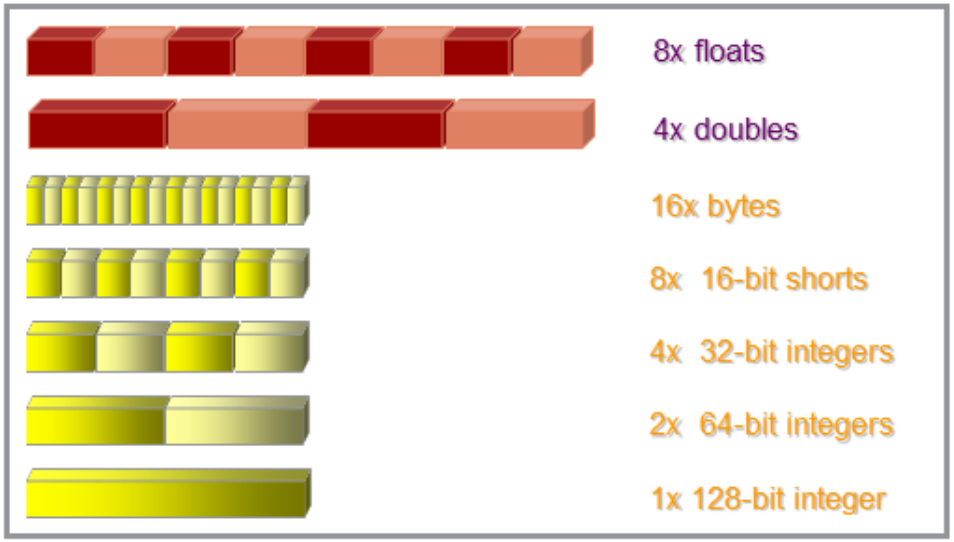
- Векторные регистры YMMX длиной 256 бит
- Набор команд FMA для выполнения векторных операций
- Новый тип кодирования инструкций

Версии AVX:

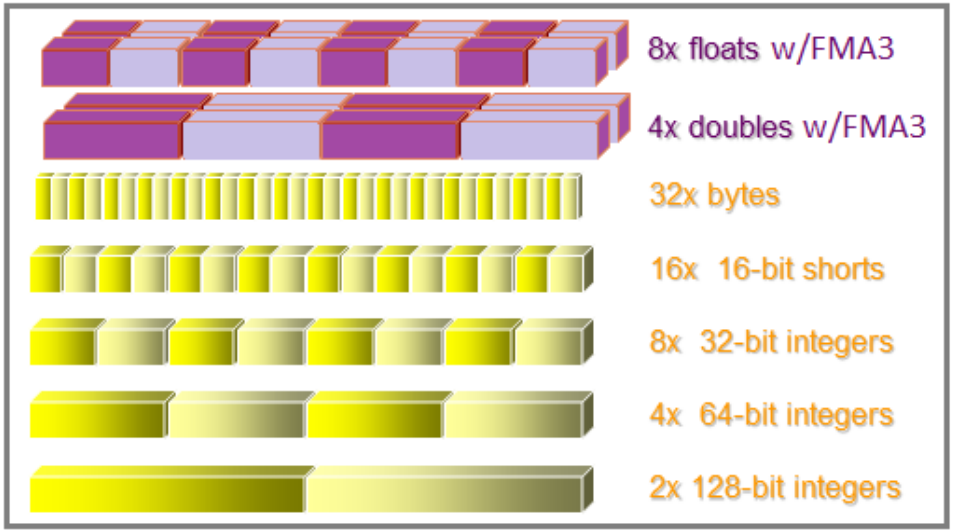
- AVX (Core 2 поколения), AVX2 (Core 4 поколения)

Векторное расширение AVX. Пакет

Intel® Advanced Vector Extensions (Intel® AVX) 1.0



Intel® AVX 2.0



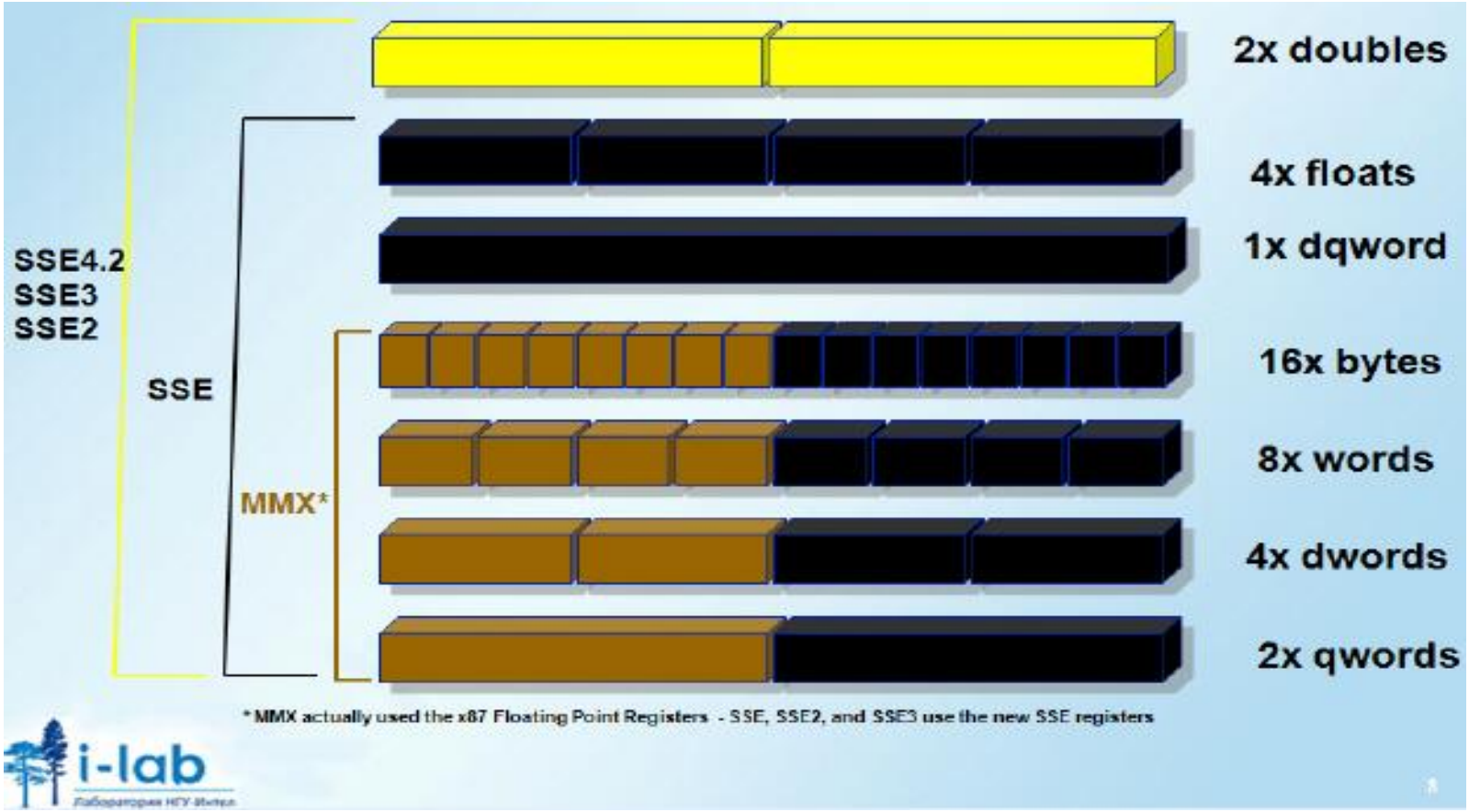
Предыдущие векторные расширения

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/SSE

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

Директивы компилятора OpenMP

Автоматическая векторизация в компиляторах

GCC
- Уровень оптимизации -O3
- Опция -ftree-vectorize

Intel
- Уровень оптимизации -O2 и -O3
- Опция -vec

clang
- Векторизация включена по умолчанию
- Выключение опцией –fno-vectorize

Пример программы

```
const int SIZE = 10240;
int a[SIZE], int b[SIZE], int c[SIZE];
int main(){
    // Инициализация массивов b и c
    ...
    for (int i = 0; i < SIZE; ++i)
        a[i] = b[i] + c[i];
    // Печатаем элемент, чтобы оптимизатор не удалил
    // цикл
    std::cout << "Last vector element =  " << a[SIZE - 1]
        << std::endl;
}
```

Результаты выполнения

С векторизацией:

```
$ g++ -O3 -std=c++11 vectorization1.cpp -o vectorization1
$ ./vectoriztion1
Last vector element =  94887
Counting time is 10
```

Без векторизации:

```
$ g++ -std=c++11 vectorization1.cpp -o vectorization1
$ ./vectorization1
Last vector element =  30892
Counting time is 39
```

Вектоизовал ли программу компилятор?

Отчет о векторизации

- Опция компилятора gcc -ftree-vectorizer-verbose=x

Уровни отчета о векторизации

- Level 0: No output at all.
- Level 1: Report vectorized loops.
- Level 2: Also report unvectorized "well-formed" loops and respective reason.
- Level 3: Also report alignment information (for "well-formed" loops).
- Level 4: Like level 3 + report for non-well-formed inner-loops.
- Level 5: Like level 3 + report for all loops.
- Level 6: Print all vectorizer dump information

Пример отчета о векторизации

Отчет о векторизации

```
$ g++ -O3  -ftree-vectorizer-verbose=1 -std=c++11
vectorization1.cpp -o vectorization1

Analyzing loop at vectorization1.cpp:29
Vectorizing loop at vectorization1.cpp:29
vectorization1.cpp:29: note: LOOP VECTORIZED.

Analyzing loop at vectorization1.cpp:23
Analyzing loop at /usr/include/c++/4.8/bits/random.tcc:902
Analyzing loop at /usr/include/c++/4.8/bits/random.tcc:925
vectorization1.cpp:16: note: vectorized 1 loops in function.
```

Какие циклы могут быть векторизованы

Заранее известно количество итераций
- Не обязательно во время компиляции, но обязательно во время выполнения
- Выход из цикла не зависит от данных

Один вход в цикл и один выход
- Не используется break, goto

Одинаковый код для всех итераций
- Не используется switch, if

Самый внутренний из вложенных циклов

Не вызываются функции
- Кроме векторизованных и inline

Маскируемые присваивания

```
void quad(int length, float *a, float *b,
float *c, float *restrict x1, float *restrict x2)
{
  for (int i=0; i<length; i++) {
    float s = b[i]*b[i] - 4*a[i]*c[i];
    if ( s >= 0 ) {
      s = sqrt(s) ;
      x2[i] = (-b[i]+s)/(2.*a[i]);
      x1[i] = (-b[i]-s)/(2.*a[i]);
    } else {
      x2[i] = 0.;
      x1[i] = 0.;
    }
  }
}
```

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

Зависимости

Возможна ли веторизация такого цикла:

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

Типы зависимостей

Read-after-write

- flow dependency
- ```
  for (j=1; j<MAX; j++)
      A[j]=A[j-1]+1;
  ```

Write-after-read

- anti-dependency
- ```
  for (j=1; j<MAX; j++)
      A[j-1]=A[j]+1;
  ```

Write-after-write

- output dependency
- ```
  for( i=0; i<N; i++)
      a[i%2] = b[i] + c[i];
  ```
  ```

Важное исключение - редукция

Скалярное произведение векторов

```
sum=0;
for (int i=1; i<MAX; i++)
    sum += a[i]*b[i]
```

Использование редукции в gcc

- По умолчанию только для целых типов
- Для векторизации чисел с плавающей точкой необходимо использовать параметр -ffast-math или -fassociative-math

Непоследовательный доступ к памяти

Доступ с шагом

```
for (int i=0; i<SIZE; i+=2)
    b[i] += a[i] * x[i];
```

Доступ по индексу

```
for (int i=0; i<SIZE; i+=2)
    b[i] += a[i] * x[index[i]];
```

Непоследовательный доступ к памяти

Доступ с шагом

```
for (int i=0; i<SIZE; i+=2)
    b[i] += a[i] * x[i];
```

Доступ по индексу

```
for (int i=0; i<SIZE; i+=2)
    b[i] += a[i] * x[index[i]];
```

```
"vectorization possible but seems inefficient"
```

Выравнивание данных

Наборы векторных инструкций включают инструкции по загрузке данных из памяти

- Загрузка «пакета» в векторный регистр
- Загрузка одного значения в векторный регистр

Для загрузки «пакета» необходимо, чтобы данные были выравнены

- SSE – 16 байт
- AVX – 32 байта

Выравнивание данных в gcc:

- `int a[SIZE] __attribute__ ((aligned (32)));`
- `posix_memalign (void **memptr, size_t align, size_t size);`

Можно ли векторизовать такой код?

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

Можно ли векторизовать такой код?

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

Указатели a, b и c могут указывать на одну область памяти

- C pointer aliasing

Решение проблемы с альясингом

```
#pragma ivdep
void add(float *a, float *b, float *c) {
    for (int i=0; i<SIZE; i++) {
        c[i] += a[i] + b[i];
    }
}
```

Решение проблемы с альясингом

```
void add(float restrict *a, float restrict *b, float
restrict *c) {
    for (int i=0; i<SIZE; i++) {
        c[i] += a[i] + b[i];
    }
}
```

Векторизация C++ vector

```
const long SIZE = 100000000;
std::vector<int> a(SIZE), b(SIZE), c(SIZE);
int main(){
    // Инициализация векторов b и c
    ...
    for (int i = 0; i < SIZE; ++i)
        a[i] = b[i] + c[i];
    std::cout << "Last vector element =  " << a[SIZE - 1]
        << std::endl;
}
```

Векторизация C++ vector

```
g++ -O3  -ftree-vectorizer-verbose=1 -std=c++11
vectorization2.cpp -o vectorization2

Analyzing loop at vectorization2.cpp:27

Vectorizing loop at vectorization2.cpp:27

vectorization2.cpp:27: note: create runtime check for data
references *_59 and *_58

vectorization2.cpp:27: note: create runtime check for data
references *_60 and *_58

vectorization2.cpp:27: note: created 2 versioning for alias
checks.

vectorization2.cpp:27: note: LOOP VECTORIZED.
```

Итоги

Векторизация позволяет значительно ускорить вашу программу

- Без векторизации невозможно полное использование вычислительной мощности современных процессоров

Последовательную программу тоже можно векторизовать

- Имеет смысл сначала векторизовать последовательную программу, а потом переходить к распараллеливанию

Используйте автоматическую векторизацию

- Не нужно менять программу при появлении нового набора инструкций

Дополнительные материалы

Auto-vectorization in GCC

http://gcc.gnu.org/projects/tree-ssa/vectorization.html

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

Вопросы?