Змістовний модуль:
СУЧАСНІ ПРОЦЕСОРИ ТА ПАРАЛЕЛІЗАЦІЯ ОБЧИСЛЕНЬ

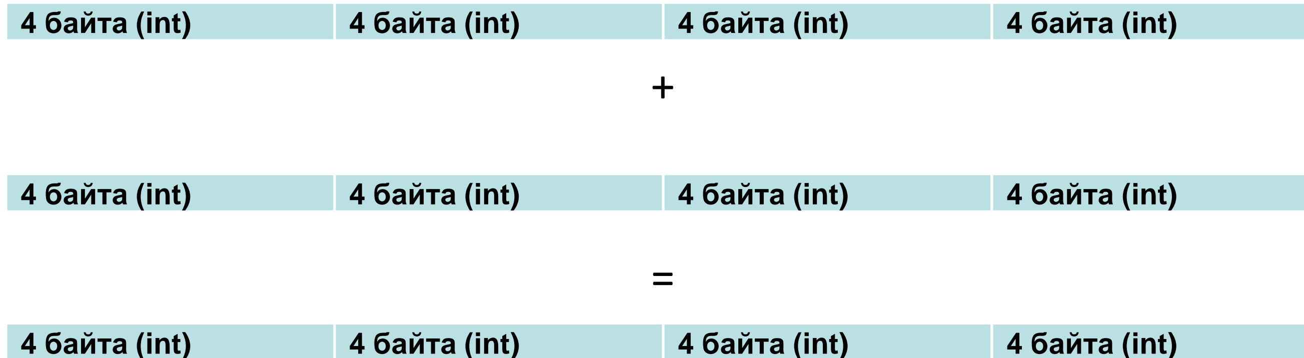
Розділ: Використання SIMD команд для паралельних обчислень

ЛЕКЦІЯ 6. SIMD КОМАНДИ. ПРОДОВЖЕННЯ

ПИТАННЯ ДЛЯ ВИВЧЕННЯ

1. Використання SSE та AVX команд для загального випадку.
2. Функції порівняння даних.
3. Функції для роботи з бітами.
4. Функції округлення.
5. Функції перемішування.

ПРИНЦИПИ РОБОТИ SIMD КОМАНД



Функції для порівняння даних з плаваючою точкою

Відносна похибка $\delta = |\Delta|/\max(a,b)$

ТИПИ ДАНИХ

Типи даних для даних з плаваючою точкою.

__m128 для чисел із плаваючою точкою звичайної точності (4 байти);

__m128d для чисел із плаваючою точкою подвійної точності (8 байтів).

```
typedef union __declspec(intrin_type) _CRT_ALIGN(16) __m128 {
 float m128_f32[4];
 unsigned __int64 m128_u64[2];
 __int8 m128_i8[16];
 __int16 m128_i16[8];
 __int32 m128_i32[4];
 __int64 m128_i64[2];
 unsigned __int8 m128_u8[16];
 unsigned __int16 m128_u16[8];
 unsigned __int32 m128_u32[4];
 } __m128; // float
typedef struct __declspec(align(16)) __m128d {
   double          m128d_f64[2];
} __m128d;//double
```

__m256 (m256_f32[0].. m256_f32[7]) __m256d(m256d_f64[0].. m256d_f64[3])

ЗАГАЛЬНИЙ ВИД ІМЕНІ ФУНКЦІЙ

<Prefix>_<Code> {p|s}{s|d},
де:

***Prefix** – _mm для SSE і _mm256 для AVX*

***Code** -* код операції, що виконується, наприклад, add, sub;

p|s показує, чи виконується функція над одним елементом (s - Single) або над всіма компонентами блока (p - Pack);

s|d задає тип даних, що обробляються (s - число звичайної точності; d - подвійної точності).

АРИФМЕТИЧНІ ОПЕРАЦІЇ. ПРИКЛАД 2

Скласти функцію для обчислення відстаней між заданою точкою й точками з масиву

Структури точки:

```
typedef struct{
        float x, y;
 }MYPOINT, *PMYPOINT;
```

АРИФМЕТИЧНІ ОПЕРАЦІЇ/ ПРИКЛАД 2

Функція для послідовного режиму і фрагмент головної програми для її використання

```
void SecDims(MYPOINT ps[], PMYPOINT p0, float Dims[], size_t n){
		const float x0 = p0->x, y0 = p0->y;
		for (size_t i = 0; i < n; i++){
				float r1 = ps[i].x - x0; float r2 = ps[i].y - y0;
				Dims[i] = sqrt(r1 * r1 + r2 * r2);
		}
}

double start, finish;
double min = DBL_MAX, diff;
for (int i = 0; i < M; i++)
				start = omp_get_wtime();
				SecDims(p, &p0, Dims1, N);
				finish = omp_get_wtime();
				diff = finish - start; if (diff < min) min = diff;
}
printf("SecDims			time = %lg\n", min);
```

АРИФМЕТИЧНІ ОПЕРАЦІЇ, ПРИКЛАД 2

***// float.* Функції з SSE. float**

```
void SSEDims(MYPOINT A[], PMYPOINT C, float Dims[], size_t n){
	const float x0 = C->x, y0 = C->y;
	__m128 m128_c = { x0, y0, x0, y0 };
	__m128 *pm128_A = (__m128 *)A;
	__m128 *pDims = (__m128 *)Dims;
	for (size_t i = 0, j = 0; i < n/2; i+= 2){
		__m128 r1 = _mm_sub_ps(pm128_A[i], m128_c);
		__m128 r2 = _mm_sub_ps(pm128_A[i + 1], m128_c);
		r1 = _mm_mul_ps(r1, r1);
		r2 = _mm_mul_ps(r2, r2);
		r1 = _mm_hadd_ps(r1, r2);
		pDims [j++] = _mm_sqrt_ps(r1);
	}
}
```

Для AVX – проблема горизонтального складання – дивись нижче!!!

АРИФМЕТИЧНІ ОПЕРАЦІЇ. ПРИКЛАД 2

Функція	Час виконання N = 1024 * 1024	Прискорення
SecDims	0.0066	1
SSEDims	0.0017	3.88
DoubleDims	?	?
SSEDoubleDims	?	?

ВИКОРИСТАННЯ SSE ТА AVX КОМАНД ДЛЯ ЗАГАЛЬНОГО ВИПАДКУ

1 Кількість даних ділиться без залишку на 4 (8) для даних типу float і на 2 (4) для даних типу double

2 Вирівнювання адреси

LOAD І STORE ФУНКЦІЇ

Призначення функцій – завантаження з пам'яті в регістр (Load) і навпаки (Store).

__m128 _mm_load_ps(float * p); void _mm_store_ps(float*,_m128);

__m128 _mm_loadu_ps(float * p); _mm_storeu_ps

__m128 _mm_load_pd(double * p); _mm_store_pd

__m128 _mm_loadu_pd(double * p); _mm_storeu_pd

__m256 _mm256 _load_ps(float * p);

__m 256 _mm256_loadu_ps(float * p); _mm256_storeu_ps

__m 256 _mm256_load_pd(double * p);

__m 256 _mm256_loadu_pd(double * p); _mm256_storeu_pd

Приклад. Реалізувати попередню задачу без вимоги вирівнювання та кратності кількості точок 4

ДАНІ ВИРІВНЯНІ. КІЛЬКІСТ КРАТНА 4

***// float.* Функції з SSE. float**

```
void SSEDims(MYPOINT A[], PMYPOINT C, float Dims[], size_t n){
	const float x0 = C->x, y0 = C->y;
	__m128 m128_c = { x0, y0, x0, y0 };
__m128 *pm128_A = (__m128 *)A, pDims = (__m128 *)Dims;
	for (size_t i = 0, j = 0; i < n/2; i+= 2){
		__m128 r1 = _mm_sub_ps(pm128_A[i], m128_c);
		__m128 r2 = _mm_sub_ps(pm128_A[i + 1], m128_c);
		r1 = _mm_mul_ps(r1, r1);
		r2 = _mm_mul_ps(r2, r2);
		r1 = _mm_hadd_ps(r1, r2);
		pDims [j++] = _mm_sqrt_ps(r1);
	}
}
```

Для AVX – проблема горизонтального складання – дивись нижче!!!

ВИРІШЕННЯ ЗАДАЧИ ДЛЯ ЗАГАЛЬНОГО ВИПАДКУ

```
void SSEDimsU (MYPOINT A[], PMYPOINT C, float Dims[], size_t n){
const float x0 = C->x, y0 = C->y;
__m128 m128_c = { x0, y0, x0, y0 };
•       __m128 *pm128_A = (__m128 *)A, pDims = (__m128 *)Dims;
size_t n_ = n / 4 * 4;
for (size_t i = 0; i < n_; i += 4){
        __m128 t1 = _mm_loadu_ps((const float*)&A[i]);
        __m128 t2 = _mm_loadu_ps((const float*)&A[i + 2]);
        __m128 r1 = _mm_sub_ps(t1, m128_c);
        __m128 r2 = _mm_sub_ps(t2, m128_c);
        r1 = _mm_mul_ps(r1, r1); r2 = _mm_mul_ps(r2, r2);
        r1 = _mm_hadd_ps(r1, r2); r1 = _mm_sqrt_ps(r1);
        _mm_storeu_ps(&Dims[i], r1);
}
for (size_t i = n_; i < n; i++){
        float r1 = A[i].x - x0; float r2 = A[i].y - y0; Dims[i] = sqrt(r1 * r1 + r2 * r2);
}} Час виконання Було 0.0017 стало 0.0018                + 5%
```

ПОРІВНЯННЯ ДАНИХ

Для функцій можна задавати умову порівняння:
Іменем функції (для SSE)
Спеціальною маскою (AVX)
Загальний вигляд імені функції з завданням умови в імені:
<Префікс>_{cmp}[<Умова>]_ {s|p}{s|d},
де:

cmp - повертають 1 у всіх бітах, якщо умова, задана в команді порівняння, виконується. Повертає 0, якщо умова не виконується;

s відповідає порівнянню одного даного, а *p* – всіх компонентів блоку; для команд типу *comi, ucomi* використовується тільки *s;*

s відповідає використанню даних зі звичайної, а *d* – з подвійною точністю.

Умови порівнювання: {eq, ne, lt, le, gt, ge, neq, nlt, nle, ngt, ge}.

Приклади функцій: *_mm_cmpeq_ss, _mm_cmpeq_ps, _mm_cmpeq_pd*

ПОРІВНЯННЯ ДАНИХ

Використання спеціального параметру (AVX)

__m256 _mm256_cmp_ **{s|p}{s|d}**(__m256 a, __m256 b, const int maska);

де:

cmp - повертають 1 у всіх бітах, якщо умова, задана в команді порівняння, виконується. Повертає 0, якщо умова не виконується;

s відповідає порівнянню одного даного, а *p* – всіх компонентів блоку; для команд типу *comi, ucomi* використовується тільки *s;*

s відповідає використанню даних зі звичайної, а *d* – з подвійною точністю.

Приклади маски:

Умови: EQ (==), NE (!=), LT (<), NLT (!<), LE (<=),NLE (<=), GT (>), NGT (!>), GE (>=), NGE (>=)

Приклади масок:

_CMP_LT_OS, _CMP_LE_OS, _CMP_NEQ_OS,…

Решта властивостей для інших функцій.

ФУНКЦІЇ ДЛЯ РОБОТИ З БІТАМИ

Код функції	Параметри	Результат	Призначення
and	a, b	r	r = a & b
or	a, b	r	r = a | b
xor	a, b	r	r = a ^ b
andnot	a, b	r	r = ~a &b

ФУНКЦІЇ ПОРІВНЯННЯ ТА РОБОТИ З БІТАМИ. Приклад 3

Вычислить значение x[i] = y [i] / z [i], если x[i]<=0, в противном случае оставить значение x[i] без изменения

Послідовний код:

```
void IfDiv(float *x, float *y, float *z, size_t n)
{

	for (size_t i = 0; i < n; ++i)
	{
		if (x[i] <=0)
		{
			x[i] = y[i] / z[i];
		}
	}
}
```

_mm_cmple_ps   _mm256_cmp_ps(…, …, **_CMP_LE_OS**);

_mm_and_ps

_mm_andnot_ps

ФУНКЦІЇ ПОРІВНЯННЯ ТА РОБОТИ З БІТАМИ. Приклад 3

Функція для SSE

```
void SSEIfDiv(float *x, float *y, float *z, size_t n)
{
	__m128 *x128 = (__m128 *)x, *y128 = (__m128 *)y, *z128 = (__m128 *)z;
	__m128 zero = _mm_setzero_ps();
	for (size_t i = 0; i < n / 4; ++i)
	{
		const __m128 r1 = _mm_div_ps(y128[i], z128[i]);
		const __m128 r2 = _mm_cmple_ps(x128[i], zero);
		const __m128 r3 = _mm_andnot_ps(r2, x128[i]);
		const __m128 r4 = _mm_and_ps(r2, r1);
			x128[i] =	_mm_or_ps(r3, r4);
	}
}
```

ФУНКЦІЇ ПОРІВНЯННЯ ТА РОБОТИ З БІТАМИ. Приклад 3

Функція для AVX

```
void AVXIfDiv(float *x, float *y, float *z, size_t n)
{
	__m256 *x256 = (__m256 *)x, *y256 = (__m256 *)y, *z256 = (__m256 *)z;
	const __m256 zero = _mm256_setzero_ps();
	for (size_t i = 0; i < n/ (sizeof(__m256)/ sizeof (float)); ++i)
	{
		const __m256 r1 = _mm256_div_ps(y256[i], z256[i]);
		const __m256 r2 = _mm256_cmp_ps(x256[i], zero, _CMP_LE_OS);
		const __m256 r3 = _mm256_andnot_ps(r2, x256[i]);
		const __m256 r4 = _mm256_and_ps(r2, r1);
		x256[i] = _mm256_or_ps(r3, r4);
	}
}
```

Результати (N = 1024 * 1024)

Функція	Час(сек)	Прискорення
IfDiv	0.0059	1
SSEIfDiv	0.0019	3.1
AVXIfDiv	0.0017	3.5

ОКРУГЛЕННЯ ЧИСЕЛ

Режим	Призначення
_MM_FROUND_TO_NEAREST_INT (0) **_MM_FROUND_NINT**	Округлення до найближчого цілого. 3.7 ≈ 4; 3.4≈3; -3.7≈-4; -3.4≈-3
_MM_FROUND_TO_NEG_INF (1)	**Округлення до найближчого меншого** 3.7 ≈ 3; 3.4≈3; -3.7≈-4; -3.4≈-4
_MM_FROUND_TO_POS_INF (2)	Округлення до найближчого більшого 3.7 ≈ 4; 3.4≈4; -3.7≈-3; -3.4≈-3
_MM_FROUND_TO_ZERO (3)	Усічення числа 3.7 ≈ 3; 3.4≈3; -3.7≈-3; -3.4≈-3
_MM_FROUND_RAISE_EXC	Виключення, якщо результат не відповідає коректному значенню
_MM_FROUND_NO_EXC	Немає виключення навіть в разі, якщо результат не відповідає коректному значенню

ОКРУГЛЕННЯ ЧИСЕЛ

Основні функції

```
__m128   _mm_round_ss(__m128 src, int mode);
__m128   _mm_round_ps(__m128 src, int mode);
__m256d  _mm256_round_pd(__m128d src, int mode);

__m256   _mm256_round_ss(__m256 src, int mode);
__m256   _mm256_round_ps(__m256 src, int mode);
__m256d  _mm256_round_pd(__m256d src, int mode);
```

Скласти функції для округлення чисел із плаваючою точкою в режимі floor

ОКРУГЛЕННЯ ЧИСЕЛ (SSE4)

Скласти функції для округлення чисел із плаваючою точкою в режимі найближчого цілого

```
void round(float *x, float *y, size_t n){
	for (size_t i = 0; i < n; i++){
		y[i] = x[i] < 0 ? x[i] - 0.5 : x[i] + 0.5; y[i] = (int)y[i];
	}
}
void SSEround(float *x, float *y, size_t n) {
	__m128 *px = (__m128 *)x, *py = (__m128 *)y;
	for (size_t i = 0; i < n / 4; ++i)
		py[i] = _mm_round_ps(px[i], 1);
}
void AVXround(float *x, float *y, size_t n){
	__m256 *px = (__m256 *)x, *py = (__m256 *)y;
	for (size_t i = 0; i < n / 8; ++i)
		py[i] = _mm256_round_ps(px[i], 1);
}
```

Часові характеристики:

Round 0.08 SSERound 0.013 AVXRound 0.013

ОКРУГЛЕННЯ ЧИСЕЛ (SSE4)

Головна програма

```
for (size_t i = 0; i < N; i++){ x[i] = (rand()  * 10.) / rand (); if (rand() & 1) x[i] = -x[i];}
min = DBL_MAX; for (int i = 0; i < 10; i++){
      start = omp_get_wtime();  round(x , y, N);  finish = omp_get_wtime(); dif = finish - start;
      if (dif < min) min = dif;
} printf("round: time = %lg\n", min);
-------------------------------------------------------------------
min = DBL_MAX; for (int i = 0; i < 10; i++){
      start = omp_get_wtime(); SSEround(x, z, N); finish = omp_get_wtime(); dif = finish - start;
      if (dif < min) min = dif;
} printf("SSEround: time = %lg %s\n", min, Compare <float>(y, z, 0.01, N) ? "Yes" : "No");
-------------------------------------------------------------------
min = DBL_MAX; for (int i = 0; i < 10; i++){
      start = omp_get_wtime(); AVXround(x, z, N); finish = omp_get_wtime(); dif = finish - start;
      if (dif < min) min = dif;
} printf("AVXround: time = %lg %s\n", min, Compare <float>(y, z, 0.01, N) ? "Yes" : "No");
```

Time ?

ФУНКЦІЇ ПЕРЕМІШУВАННЯ

- Функції дозволяють вибрати компоненти із заданими номерами з двох блоків відповідно до заданої маски. Результат формується з окремих компонентів першого та другого блоків
- #define _MM_SHUFFLE(z, y, x, w) \ ((z<<6) | (y<<4) | (x<<2) | w)

r[0] = a[w]; r [1] = a [x]; r[2] = b[y]; r[3] = b [z]

#define _MM_*SHUFFLE2(x, y) ((x<<1) | y)*

r[0] = a[y]; r [1] = b [x]

_mm_shuffle_ps

_mm_shuffle_pd

ФУНКЦІЇ ПЕРЕМІШУВАННЯ

Приклад

Обчислити добуток для комплексних чисел c = a * b;

```
typedef struct COMPLEX{
    float re, im;
}*PCOMPLEX;

void cmul(PCOMPLEX a, PCOMPLEX b, PCOMPLEX c, size_t n)
{
    for (size_t i = 0; i < n; i++){
            c[i].re = a[i].re * b[i].re - a[i].im * b[i].im;
            c[i].im = a[i].re * b[i].im + a[i].im * b[i].re;
    }
}
```

ФУНКЦІЇ ПЕРЕМІШУВАННЯ

```
void SSE_cmul(PCOMPLEX a, PCOMPLEX b, PCOMPLEX c, size_t n){
    __m128 *a128 = (__m128 *) a, *b128 = (__m128 *)b, *c128 = (__m128 *)c;
    for (size_t i = 0; i < n / 2; i++){
        __m128 r_1 = _mm_mul_ps(_mm_shuffle_ps(b128[i], b128[i], 0xA0), a128[i]);
        __m128 r_2 = _mm_mul_ps(_mm_shuffle_ps(b128[i], b128[i], 0xF5), a128[i]);
        c128[i] = _mm_addsub_ps(r_1, _mm_shuffle_ps(r_2, r_2, 0xB1));
    }
}
void AVX_cmul(PCOMPLEX a, PCOMPLEX b, PCOMPLEX c, size_t n){
    __m256 *a256 = (__m256 *) a, *b256 = (__m256 *)b, *c256 = (__m256 *)c;
    for (size_t i = 0; i < n/4; i++){
        __m256 r_1 = _mm256_mul_ps(_mm256_shuffle_ps(b256[i], b256[i], 0xA0),
    a256[i]);
        __m256 r_2 = _mm256_mul_ps(_mm256_shuffle_ps(b256[i], b256[i], 0xF5),
    a256[i]);
    c256[i] = _mm256_addsub_ps(r_1, _mm256_shuffle_ps(r_2, r_2, 0xB1));}}
```

ДОДАТКОВІ ФУНКЦІЇ

Функція для додавання добутків деяких компонентів полів:

__m128 _mm_dp_ps(__m128 a, __m128 b, const int mask);

__m256 _mm256_dp_ps(__m256 a, __m256 b, const int maska);

Обчислює суму добутків.

Які добутки обчислюються задає старші 4 біта маски: 1 – компонент блоку перемножується. Якщо треба обчислювати добутки усіх компонентів 1111

Молодші 4 біта задають поля, куди результат після обчислення суми

Решта полів результату – 0.

Для AVX довжина маски залишилася без змін, тому маска задає зразу 2 поля.

ДОДАТКОВІ ФУНКЦІЇ

Приклад 1

```
a128 = _mm_setr_ps(1., 2., 3., 4.); b128 = _mm_setr_ps(8., 9., 10., 11.);
c128 = _mm_dp_ps(a128, b128, 0xFA);
for (int i = 0; i < 4; i++)
printf("%g\n", c128.m128_f32[i]);
```

Результат

….

Приклад 2

```
a256 = _mm256_setr_ps(1., 1., 2., 3., 4., 5., 6., 7.);
b256 = _mm256_setr_ps(8., 9., 10., 11., 12., 13., 14., 15.);
c256 = _mm256_dp_ps(a256, b256, 0x2A);
for (int i = 0; i < 8; i++)
printf("%g\n", c256.m256_f32[i]);
```

ДОДАТКОВІ ФУНКЦІЇ

Приклад 1

```
a128 = _mm_setr_ps(1., 2., 3., 4.); b128 = _mm_setr_ps(8., 9., 10., 11.);
c128 = _mm_dp_ps(a128, b128, 0xFA);
for (int i = 0; i < 4; i++)
printf("%g\n", c128.m128_f32[i]);
```

Результат

Маска : 1111

1 * 8 + 2 * 9 + 3 * 10 + 4 * 11 = 100

C128[0] = C128[2] = 100; C128[1] = C128[3] = 0;

Приклад 2

```
a256 = _mm256_setr_ps(1., 1., 2., 3., 4., 5., 6., 7.);
b256 = _mm256_setr_ps(8., 9., 10., 11., 12., 13., 14., 15.);
c256 = _mm256_dp_ps(a256, b256, 0x3A);
for (int i = 0; i < 8; i++) printf("%g\n", c256.m256_f32[i]);
```

Маска 0011. поля 0 и 1 ; 5 и 6 (1 * 8 + 1 * 9) и 12 * 4 + 13 * 5

Результаты записывают в поля c256 [1] = c256 [3] = 17; c256 [5] = c256 [7] =113. решта - 0

ДОДАТКОВІ ФУНКЦІЇ

Приклад 3

Скласти функцію для обчислення скалярного добутку з використанням звичайних обчислень, SSE, AVX звичайних та додаткових операцій

ДОДАТКОВІ ФУНКЦІЇ

Приклад 3

Скласти функцію для обчислення скалярного добутку з використанням звичайних обчислень, SSE, AVX звичайних та додаткових операцій

```
float smSSE(float *x, float *y, size_t n){
	__m128 *px = (__m128 *)x, *py = (__m128 *)y, s128 = _mm_setzero_ps();
	float *s = (float*)&s128;
	for (size_t i = 0; i < n / 4; i++)
			s128 = _mm_add_ps(s128, _mm_mul_ps(px[i], py[i]));
	s128 = _mm_hadd_ps(s128, s128);
	return s[0] + s[1];
}
float smAVX1(float *x, float *y, size_t n){
	__m256 *px = (__m256 *)x, *py = (__m256 *)y, s256 = _mm256_setzero_ps();
	float *s = (float*)&s256;
	for (size_t i = 0; i < n / 8; i++)
			s256 = _mm256_add_ps(s256, _mm256_mul_ps(px[i], py[i]));
	s256 = _mm256_hadd_ps(s256, s256); s256 = _mm256_hadd_ps(s256, s256);
	return s[0] + s[4];
//dest [0]=s1[0]+s1[1]; dest[1]=s1[2]+s1[3]; dest[2]=s2[0]+s2[1]; dest[3]=s2[2]+s2[3];
//dest [4]=s1[4]+s1[5]; dest[5]=s1[6]+s1[7]; dest[6]=s2[4]+s2[5]; dest[7]=s2[6]+s2[7];
}
```

ДОДАТКОВІ ФУНКЦІЇ

Приклад 3

Скласти функцію для обчислення скалярного добутку з використанням звичайних обчислень, SSE, AVX звичайних та додаткових операцій

```
// Функція з використанням комплексної AVX операції _mm256_dp_ps
float smAVX2(float *x, float *y, size_t n){
    __m256 *px = (__m256 *)x, *py = (__m256 *)y;
    __m256 s256 = _mm256_setzero_ps();
    float *s = (float*)&s256;
    for (size_t i = 0; i < n / 8; i++)
        s256 = _mm256_add_ps(s256, _mm256_dp_ps(px[i], py[i], 0xF1));
    return s[0] + s[4];
}
```

Часові характеристики

Sm	0.0028	smSSE	0.0012	smAVX1	0.00071	smAVX2	0.00071

ВИСНОВКИ

- Найбільше розповсюдження мають SSE та AVX команди, які реалізовані для усіх сучасних процесорів загального призначення.
- В лекції досліджені деякі типи цих команд.
- Використання команд вимагає, щоб адреси початку масивів були вирівняні на границю даних типу __m128 (__m256), тобто ділилися на 16 (32).
- Порівняння різних типів команд показує, що є команди, які дозволяють отримати прискорення дуже значне (наприклад, функції, пов'язані з обчисленням кореня квадратного). На жаль, є команди, які не дають прискорення. Таким чином, перед використанням має сенс досліджувати ефективність команд для конкретних масивів.
- Сучасні процесори мають декілька версій SSE команд. Перед використанням старших версій необхідно обов'язково перевіряти можливість їх використання. Як це робити – буде розглянуто нижче.
- Далі будуть розглянуті функції для роботи з цілими числами.

ПИТАННЯ ДЛЯ САМОСТІЙНОГО ВИВЧЕННЯ

1. Вивчить команди типу MMX, 3DNow!, порівняйте їх ефективність з відповідними командами SSE, зробіть висновки.
2. Дослідіть можливість використання SSE операцій для виконання дій для масивів комплексних чисел та перевірте їх ефективність.
3. Використовуючи MSDN знайдіть класи функцій для чисел з плаваючою точкою, які не розглянуті в курсі, вивчіть їх та перевірте ефективність їх використання.

МАТЕРІАЛИ ДЛЯ ЕКСПРЕС-КОНТРОЛЮ

1. Як вирівняти адресу масиву на задану границю?
2. Що робити, якщо довжина масиву не кратна розміру блоку?
3. Складіть програму для виконання усіх арифметичних операцій без використання та з використанням SSE та дослідіть залежність прискорення від операції, типу даних, розміру масивів.
4. Знайдіть серед команд команди для перетворення даних та дослідіть їх ефективність.
5. Вивчіть макроси для округлення даних та наведіть приклади їх використання.