

**Звіт з лабораторної роботи  
за дисципліною "Архітектура і програмне  
забезпечення обчислювальних систем"  
студента групи ПА-17-1  
Панасенка Егора Сергійовича  
Кафедра комп'ютерних технологій, фпм, дну  
2017/2018 навч.р.**

1. Постановка задачі:

1. Реализовать процедуру умножения квадратных матриц (размером кратным четырём) без использования специальных расширений и с использованием расширений SSE, сравнить время выполнения этих реализаций.
2. В соответствии с вариантом задания реализовать матрично-векторную (с одинаковым размером матриц и векторов кратным четырём) процедуру с использованием расширений SSE.
3. С использованием инструкции `cuid` определить наличие расширения SSE.

2. Опис коду:

1. У програмі є багато функцій для роботи з матрицями, векторами та числами, причому до кожної функції є подібна їй з використанням SSE:
  1. `float rand_f (int min_number, char max_number)`  
`__m128 rand_f_sse (int min_number, char max_number)`
    1. Виводить випадкове число у заданому діапазоні
    2. Аргументи:
      1. `min_number` – мінімальне число у діапазоні чисел
      2. `max_number` – максимальне число у діапазоні чисел
  2. `float ** create_matrix (int size, int min_number, char max_number)`  
`__m128 ** create_matrix_sse (int size, int min_number, char max_number)`
    1. Створює та заповнює матрицю випадковими числами, якщо це потрібно (`min_number != max_number`)
    2. Аргументи:
      1. `size` – розмір матриці

- 2. min\_number – мінімальне число у діапазоні чисел
- 3. max\_number – максимальне число у діапазоні чисел
- 3. void print\_matrix (FILE \* output, float \*\* a ,int size, const char \* name)
  - void print\_matrix\_sse (FILE \* output, \_\_m128 \*\* a ,int size, const char \* name)
  - 1. Виводить у потік матрицю
  - 2. Аргументи:
    - 1. output – потік
    - 2. a – матриця
    - 3. size – розмір матриці
    - 4. name – назва матриці
- 4. void free\_m (float \*\* a, int size)
  - void free\_m\_sse (\_\_m128 \*\* a, int size)
  - 1. Звільнює пам'ять матриці
  - 2. Аргументи:
    - 1. a – матриця
    - 2. size – розмір матриці
- 5. float \* create\_vector (int size, int min\_number, char max\_number)
  - \_\_m128 \* create\_vector\_sse (int size, int min\_number, char max\_number)
  - 1. Створює та заповнює вектор випадковими числами, якщо це потрібно (min\_number != max\_number)
  - 2. Аргументи:
    - 1. size – розмір вектора
    - 2. min\_number – мінімальне число у діапазоні чисел
    - 3. max\_number – максимальне число у діапазоні чисел
- 6. void print\_vector (FILE \* output, float \* a ,int size, const char \* name)
  - void print\_vector\_sse (FILE \* output, \_\_m128 \* a ,int size, const char \* name)
  - 1. Виводить у потік вектор
  - 2. Аргументи:

1. output – потік
  2. a – вектор
  3. size – розмір матриці
  4. name – назва матриці
7. float \*\* mul\_mat (float \*\* a, float \*\* b, int size)  
\_\_m128 \*\* mul\_mat\_sse (\_\_m128 \*\* a, \_\_m128 \*\* b, int size)
1. Перемножує дві матриці
  2. Аргументи:
    1. a і b - матриці
    2. size – розмір матриць
8. float \*\* mul\_num\_mat (float a, float \*\* b, int size)  
\_\_m128 \*\* mul\_num\_mat\_sse (\_\_m128 a, \_\_m128 \*\* b, int size)
1. Перемножує число на матрицю
  2. Аргументи:
    1. a – число
    2. b – матриця
    3. size – розмір матриці
9. float \* mul\_mat\_vec (float \*\* a, float \* b, int size)  
\_\_m128 \* mul\_mat\_vec\_sse (\_\_m128 \*\* a, \_\_m128 \* b, int size)
1. Перемножує матрицю на вектор
  2. Аргументи:
    1. a – матриця
    2. b – вектор
    3. size – розмір матриці
10. float \*\* add\_mat (float \*\* a, float \*\* b, int size)  
\_\_m128 \*\* add\_mat\_sse (\_\_m128 \*\* a, \_\_m128 \*\* b, int size)
1. Додає дві матриці
  2. Аргументи:
    1. a і b - матриці
    3. size – розмір матриць
11. float \* add\_vec (float \* a, float \* b, int size)

```
__m128 * add_vec_sse (__m128 * a, __m128 * b, int size)
```

1. Додає дві матриці

2. Аргументи:

1. a і b - вектори

2. size – розмір матриць

2. В головній функції виконуються такі дії

1. Ініціювання усіх необхідних змінних

2. Створення двох випадкових матриць

3. Перемноження цих матриць без SSE та з SSE

4. Показ на екран цих матриць

5. Створення випадкової матриці, перемноження матриці на 4 без SSE та з SSE та показ на екран цих матриць

6. Створення випадкових матриці та вектор, перемноження матриці на вектор без SSE та з SSE на 4 та показ на екран цих матриці та отриманих векторів

7. Створення двох випадкових матриць, додавання цих матриць без SSE та з SSE, показ на екран цих матриць

8. Створення двох випадкових векторів, додавання цих векторів без SSE та з SSE, показ на екран цих векторів

9. Створення необхідних випадкових даних та обчислення формули без SSE та з SSE, показ на екран початкових та отриманих даних

10. Збір статистики швидкостей без SSE та з SSE при перемноженні матриць

11. Збір статистики швидкостей без SSE та з SSE при обчисленні формули

3. Опис результатів:

1. Завдяки тому, що ми за допомогою SSE, об'єднуємо 4 числа типу float (хоча ми можемо брати не тільки float), ми можемо досягти збільшення швидкості деяких дій до 4 раз, на графіку ми бачимо що відношення швидкостей варіюється від 2 до 3 разів, це завдяки тому що деякі стандартні операції, які не можна обробити за допомогою SSE, уже ж таки виконується, тому дійсно збільшити швидкість у 4 рази неможливо.

## 4. Інформація про комп'ютер:

**INTEL** (R) Core(TM) i3-3217U CPU  
1795.928 MHz  
TEMP: 56°C CRIT: 105°C  
Family: 6 | Model: 58 | Stepping: 9 | Cores: 4 | Phy ID: 0  
Address sizes: 36 bits physical, 48 bits virtual

Flags: MMX, SSE, SSE2, XD-Bit, SSE3, SSSE3, SSE4.1, SSE4.2, **Hyper-Threading**

**Power management:**  
Data from the manufacturer: Bogomips: 3591.85

Integrated GPU: Intel HD 4000  
Codename: Ivy Bridge  
TDP: 17 W  
Package: FC-BGA12F  
PQS: 22 nm  
Socket: Intel BGA1023  
Transistors: unknown  
Multiplier: 18.0x  
Die size: 118 mm²  
Voltage: unknown

**Cache L1 32K Data WOA: 8 CLS: 64 NOS: 64**  
**Cache L1 32K Instruction WOA: 8 CLS: 64 NOS: 64**  
**Cache L2 256K Unified WOA: 8 CLS: 64 NOS: 512**  
**Cache L3 3072K Unified WOA: 12 CLS: 64 NOS: 4096**

CPUINFO CPUID FLAGS 5%  
7.6.0 cpu3 Close

CPU Vendor: GenuineIntel  
CPU Codename: Sandy Bridge (Core i3)  
CPU Brand: Intel(R) Core(TM) i3-3217U CPU @ 1.80GHz  
CPU Vendor: Intel CPU  
CPU Family: 6  
CPU Model: 10  
CPU Stepping: 9  
CPU Ext family: 6  
CPU Ext model: 58  
CPU Num cores: 2  
CPU Num logical cpus: 4  
CPU Total logical cpus: 4  
L1 Data Cache: 32 KB, Assoc: 8-way, Cacheline: 64 byte  
L1 Instruction Cache: 32 KB  
L2 Cache: 256 KB, Assoc: 8-way, Cacheline: 64 byte  
L3 Cache: 3072 KB, Assoc: 12-way, Cacheline: 64 byte  
L4 Cache: 1 KB, Assoc: undetermined, Cacheline: undetermined  
SSE Size: 128 bit (non-authoritative)  
Clock: 1795 Mhz, By OS: 1795 Mhz, By IC: 1773 Mhz, Measure: 1795 Mhz  
CPU Mark TSC: 5104, Sys Clock: 2

CPUINFO CPUID FLAGS 2%  
7.6.0 Close

**Intel Corporation 3rd Gen Core processor Graphics Controller**

OpenGL Vendor: Intel Open Source Technology Center  
OpenGL Renderer: Mesa DRI Intel(R) Ivybridge Mobile  
OpenGL Version: 3.0 Mesa 17.2.8  
Client glx vendor: Mesa Project and SGI  
Client glx version: 1.4  
Server glx vendor: SGI  
Server glx version: 1.4  
GLX version: 1.4  
OpenGL ES PVS: OpenGL ES 3.0 Mesa 17.2.8  
OpenGL ES PSLVS: OpenGL ES GLSL ES 3.00

Kernel driver in use: i915  
Total Memory prefetchable: 4M -  
Total Memory non-prefetchable: 256M -  
Present resolution: 1366x768 pixels (361x203 millimeters)  
minimum 8 x 8, current 1366 x 768, maximum 32767 x 32767

EDID  
Manufacturer: AUO Model 47ec Serial Number 0  
Monitor name: AUO B156XW00DPMS: On Gamma: 2.20  
Status: connected Enabled: enabled  
Maximum image size: 34 cm x 19 cm Serial number:

7.6.0 Force card0-eDP-1 Close

**Board**  
Board Vendor: Acer  
Board Version: V2.06  
Board Name: EA50\_CX  
Board Asset Tag: Type2 - Board Asset Tag

**Bios**  
Bios Vendor: Insyde Corp.  
Bios Version: V2.06  
Bios Date: 10/08/2013

**Chassis**  
Chassis Vendor: Insyde Corp.  
Chassis Version: V2.06  
Chassis Type: Acer Asset Tag String  
Chassis Asset Tag: Aspire E1-570G

**Product**  
Product Name: Aspire E1-570G  
Product Version: V2.06

7.6.0 Close

HDA-Intel - HDA Intel PCH input10  
/proc/asound/PCH/codemc#0.Codec: Realtek HDA Intel PCH Headphone  
ALSA Driver Version k4.13.0-generic.

Node 0x20 [Vendor Defined Widget] wcaps 0xf00040: Mono  
Processing caps: benign=0, ncoeff=117  
Node 0x21 [Pin Complex] wcaps 0x40058d: Stereo Amp-Out  
Control: name="Headphone Playback Switch", index=0, device=0  
ControlAmp: chs=3, dir=Out, id=0, ofs=0  
Amp-Out caps: ofs=0x00, nsteps=0x00, stepsize=0x00, mute=1  
Amp-Out vals: [0x00 0x00]  
Pin cap 0x0000001c: OUT HP Detect  
Pin Default 0x0321101f: [Jack] HP Out at Ext Left  
Conn = 1/8, Color = Black  
DefAssociation = 0x1, Sequence = 0xf  
Pin-ctls: 0xc0: OUT HP  
Unsolicted: tag=01, enabled=1  
Power states: D0 D1 D2 D3 EPSS  
Power: setting=D3, actual=D3  
Connection: 2  
0xc0 0x0d4  
Node 0x22 [Audio Mixer] wcaps 0x20010b: Stereo Amp-In  
Amp-In caps: ofs=0x00, nsteps=0x00, stepsize=0x00, mute=1  
Amp-In vals: [0x80 0x80] [0x80 0x80] [0x80 0x80] [0x80 0x80] [0x80 0x80] [0x80 0x80]  
Connection: 6  
0x18 0x19 0x1a 0x1b 0x1d 0x0b  
Node 0x23 [Audio Mixer] wcaps 0x20010b: Stereo Amp-In  
Amp-In caps: ofs=0x00, nsteps=0x00, stepsize=0x00, mute=1  
Amp-In vals: [0x80 0x80] [0x80 0x80] [0x80 0x80] [0x80 0x80] [0x80 0x80] [0x80 0x80]  
Connection: 6  
0x18 0x19 0x1a 0x1b 0x1d 0x0b

7.6.0 Close

**500.1 GB** Vendor: ATA Device rev: SDM1  
Model: ST500LT012-9WS14 Filesystem: SWAP  
State: running Rotational: 1 Removable: No Port: SATA0

Point:  
UUID: ZadeBb72-2cd2-4ad1-90b6-94b12750e8d7  
Width:  
0%

Disk size: 465.7 GB Part size: 954 MB Free: 954

Modaliases: scsi:t-0x00 HW SATA SPD Limit: 6.0 Gbps  
Add random: 1 SATA SPD: 3.0 Gbps  
Physical block size: 4096 SATA SPD Limit: 6.0 Gbps  
Logical block size: 512 Range: 16 Ext range: 256  
Minimum IO size: 4096 Max sectors: Max sectors kb: 1280  
Optimal IO size: 0 Max HW sectors kb: 32767  
HW sector size: 512 Max segments: 128 Max segment size: 65536  
Rq affinity: 1 Discard max bytes: 0 Discard granularity: 0  
IO stats: 1 Discard zeroes data: 0  
Read ahead kb: 128 BDI stable pages required: 0  
Nr requests: 128 BDI min\_ratio: 0 ...Scheduler... cfq  
Nomerges: 0 BDI max\_ratio: 100

7.6.0 sda sda4 Close

**System**  
Distro Vendor: Ubuntu  
Distro Release: 16.04  
Desktop environment: LXDE  
Window manager: OpenBox  
Window manager Theme: Ubuntu-small  
GTK+ 2 Theme: Ubuntu-default  
GTK+ 3 Theme: Not Found  
GTK Font: Comic Sans Free Pro 8  
GTK Icons: Ubuntu  
System SHELL: /bin/sh

**Library**  
GCC Version: 5.4.0 Linux C Library:  
Hostname: gaura GNU Make Version: 4.1  
Arch: x86\_64 PPP: 2.4.7  
Timezone: Europe/Zaporozhye Dynamic linker (ldd): 2.23  
X.Org Version: 1.19.5 Net-tools: 1.60  
GLX Version: 1.4 Wireless-tools: 30  
Binutils 2.26.1  
Default Display Manager: /usr/sbin/lightdm

Uptime: 03:10:48

7.6.0 Close

**Kernel**  
Machine: x86\_64  
OS Type: Linux  
Hostname: gaura  
OS Release: 4.13.0-37-generic  
Build Date: #42~16.04.1-Ubuntu SMP Wed Mar 7 16:03:28 UTC 2018

Installed Kernels: 2  
vmlinuz-4.13.0-36-generic  
vmlinuz-4.13.0-37-generic

BOOT\_IMAGE=/boot/vmlinuz-4.13.0-37-generic root=UUID=bce79534-7b8a-45bc-b12e-964d7068ccda ro quiet splash rdblacklist=nouveau resume=/dev/disk/by-uuid/ZadeBb72-2cd2-4ad1-90b6-94b12750e8d7 vt.handoff=7

Show modules Close

**Memory**  
Standard Info Serial Presence Detect (Beta)

Mem used: 1493.332 MB  
38%  
Swap used: 0 MB  
0%

Heap size is 6,160,384 bytes.

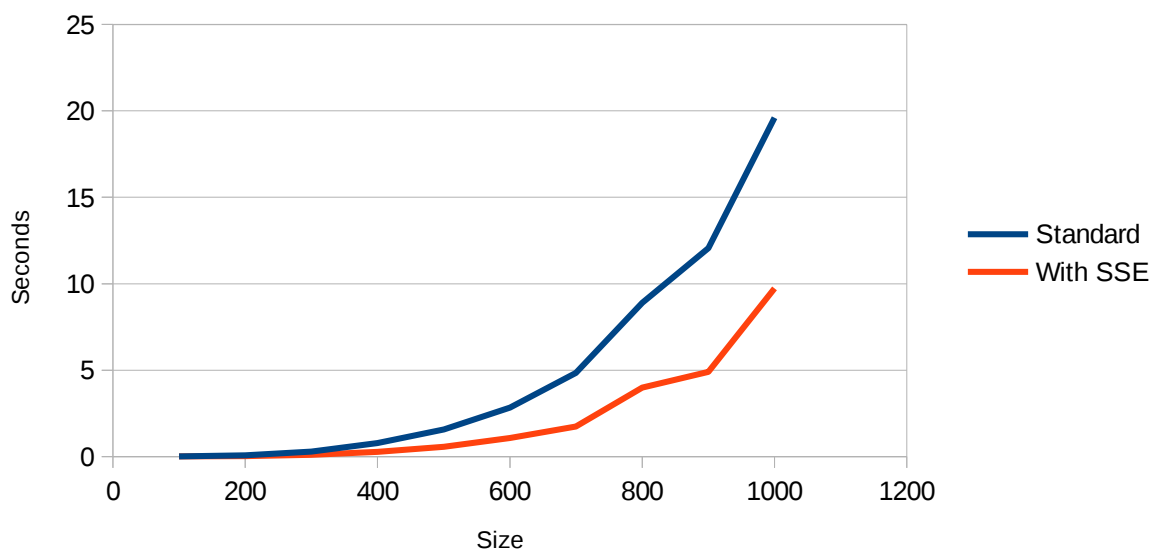
Mem Total: 3929.476MB  
Mem Free: 520.908MB  
Mem Available: 1843.056MB  
Buffers: 117.828MB  
Cached: 1797.408MB  
Swap Free: 976.892MB Swap Cached: 0MB  
Swap Total: 976.892MB Active: 2136.808MB

7.6.0 Close

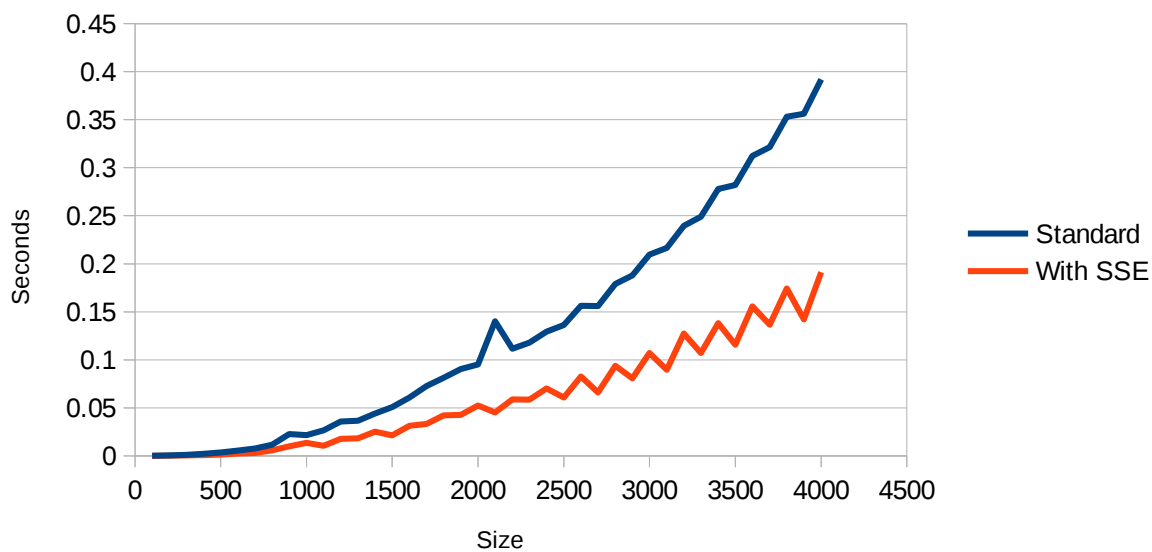
## 5. Таблиці та графіки:

Multiply 2 matrices			Compute formula		
Size	Standard	With SSE	Size	Standard	With SSE
100	0.011975	0.006474	100	0.000125	0.000048
200	0.082635	0.03053	200	0.000478	0.000186
300	0.284289	0.108448	300	0.001148	0.000434
400	0.788482	0.281642	400	0.002104	0.000876
500	1.571325	0.561944	500	0.003523	0.001469
600	2.836564	1.074336	600	0.00552	0.002327
700	4.84193	1.740416	700	0.007814	0.002868
800	8.895281	3.985972	800	0.011509	0.005718
900	12.055253	4.898085	900	0.022671	0.010001
1000	19.605669	9.738386	1000	0.021528	0.013402
			1100	0.02665	0.010397
			1200	0.035869	0.017596
			1300	0.036739	0.018358
			1400	0.044098	0.02525
			1500	0.050716	0.02134
			1600	0.060784	0.031195
			1700	0.072656	0.033197
			1800	0.081367	0.042162
			1900	0.09042	0.042685
			2000	0.095099	0.05231
			2100	0.140222	0.04519
			2200	0.111707	0.058894
			2300	0.1179	0.058572
			2400	0.129483	0.070316
			2500	0.136429	0.060709
			2600	0.156198	0.082704
			2700	0.155931	0.066116
			2800	0.178979	0.093705
			2900	0.187897	0.080826
			3000	0.209602	0.107083
			3100	0.216417	0.089779
			3200	0.239477	0.12752
			3300	0.249023	0.107179
			3400	0.278025	0.138099
			3500	0.281982	0.115805
			3600	0.312394	0.155651
			3700	0.321608	0.136493
			3800	0.353174	0.174369
			3900	0.356118	0.142215
			4000	0.391971	0.191166

### Multiply 2 matrices



### Compute formula



### 6. Коди програм, скриптів:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <xmmintrin.h>

float rand_f (int min_number, char max_number) {
    return rand()%(max_number-min_number+1)+min_number;
}
```

```

__m128 rand_f_sse (int min_number, char max_number) {
    return _mm_set_ps(
        rand_f(min_number,max_number),
        rand_f(min_number,max_number),
        rand_f(min_number,max_number),
        rand_f(min_number,max_number)
    );
}

float ** create_matrix (int size, int min_number, char max_number) {
    float ** a = (float**) _mm_malloc (size*sizeof(float*),16);
    for (int i=0;i<size;i++) {
        a[i] = (float*) _mm_malloc (size*sizeof(float),16);
        for (int j=0;j<size;j++)
            if (min_number==max_number) a[i][j] = min_number;
            else a[i][j] = rand_f(min_number,max_number);
    }
    return a;
}

__m128 ** create_matrix_sse (int size, int min_number, char max_number) {
    int s = size/4;
    __m128 ** a = (__m128**) _mm_malloc (size*sizeof(float*),16);
    for (int i=0;i<size;i++) {
        a[i] = (__m128*) _mm_malloc (size*sizeof(float),16);
        for (int j=0;j<s;j++)
            if (min_number==max_number) a[i][j] = _mm_set1_ps(min_number);
            else a[i][j] = rand_f_sse(min_number,max_number);
    }
    return a;
}

void print_matrix (FILE * output, float ** a ,int size, const char * name) {
    fprintf(output,"Matrix %s:\n",name);
    for (int i=0;i<size;i++) {
        for (int j=0;j<size;j++) {
            fprintf(output,"%5i ",(int)a[i][j]);
        }
        fprintf(output,"\n");
    }
    fprintf(output,"\n");
}

void print_matrix_sse (FILE * output, __m128 ** a ,int size, const char * name) {
    fprintf(output,"Matrix %s with SSE:\n",name);
    for (int i=0;i<size;i++) {
        for (int j=0;j<size;j++) {
            fprintf(output,"%5i ",(int)*((float*)&a[i][j/4])+j%4));
        }
        fprintf(output,"\n");
    }
    fprintf(output,"\n");
}

void free_m (float ** a, int size) {
    for (int i=0;i<size;i++) _mm_free(a[i]);
    _mm_free(a);
}

```



```

void free_m_sse (__m128 ** a, int size) {
    for (int i=0;i<size/4;i++) _mm_free(a[i]);
    _mm_free(a);
}

float * create_vector (int size, int min_number, char max_number) {
    float * a = (float*) _mm_malloc (size*sizeof(float),16);
    for (int i=0;i<size;i++) {
        if (min_number==max_number) a[i] = min_number;
        else a[i] = rand_f(min_number,max_number);
    }
    return a;
}

__m128 * create_vector_sse (int size, int min_number, char max_number) {
    int s = size/4;
    __m128 * a = (__m128*) _mm_malloc (size*sizeof(float),16);
    for (int i=0;i<s;i++) {
        if (min_number==max_number) a[i] = _mm_set1_ps(min_number);
        else a[i] = rand_f_sse(min_number,max_number);
    }
    return a;
}

void print_vector (FILE * output, float * a ,int size, const char * name) {
    fprintf(output,"Vector %s:\n",name);
    for (int i=0;i<size;i++)
        fprintf(output,"%5i ",(int)a[i]);
    fprintf(output,"\n\n");
}

void print_vector_sse (FILE * output, __m128 * a ,int size, const char * name) {
    fprintf(output,"Vector %s with SSE:\n",name);
    for (int i=0;i<size;i++) {
        fprintf(output,"%5i ",(int)*((float*)&a[i/4]+i%4));
    }
    fprintf(output,"\n\n");
}

float ** mul_mat (float ** a, float ** b, int size) {
    float ** c = create_matrix(size,0,0);
    for (int i=0;i<size;i++) {
        for (int j=0;j<size;j++) {
            c[i][j] = 0;
            for (int k=0;k<size;k++) {
                c[i][j]+=a[i][k]*b[k][j];
            }
        }
    }
    return c;
}

__m128 ** mul_mat_sse (__m128 ** a, __m128 ** b, int size) {
    __m128 ** c = create_matrix_sse(size,0,0);
    int s = size/4;
    for (int i=0;i<size;i++) {

```

```

        for (int j=0;j<s;j++) {
            c[i][j] = _mm_set1_ps(0);
            for (int k=0;k<size;k++) {
                c[i][j] = _mm_add_ps(c[i][j],
                    _mm_mul_ps(
                        _mm_set1_ps(*((float*)&a[i][k/4])+k%4))
                        , b[k][j]));
            }
        }
    }
    return c;
}

float ** mul_num_mat (float a, float ** b, int size) {
    float ** c = (float**) _mm_malloc (size*sizeof(float*),16);
    for (int i=0;i<size;i++) {
        c[i] = (float*) _mm_malloc (size*sizeof(float),16);
        for (int j=0;j<size;j++)
            c[i][j]=a*b[i][j];
    }
    return c;
}

__m128 ** mul_num_mat_sse (__m128 a, __m128 ** b, int size) {
    int s = size/4;
    __m128 ** c = (__m128**) _mm_malloc (size*sizeof(__m128*),16);
    for (int i=0;i<size;i++) {
        c[i] = (__m128*) _mm_malloc (s*sizeof(__m128),16);
        for (int j=0;j<s;j++)
            c[i][j] = _mm_mul_ps(a, b[i][j]);
    }
    return c;
}

float * mul_mat_vec (float ** a, float * b, int size) {
    float * c = create_vector(size,0,0);
    for (int i=0;i<size;i++)
        for (int j=0;j<size;j++)
            c[i]+=a[j][i]*b[i];
    return c;
}

__m128 * mul_mat_vec_sse (__m128 ** a, __m128 * b, int size) {
    int s = size/4;
    __m128 * c = create_vector_sse(size,0,0);
    for (int i=0;i<s;i++)
        for (int j=0;j<size;j++)
            c[i] = _mm_add_ps(c[i],
                _mm_mul_ps(a[j][i], b[i]));
    return c;
}

float ** add_mat (float ** a, float ** b, int size) {
    float ** c = (float**) _mm_malloc (size*sizeof(float*),16);
    for (int i=0;i<size;i++) {
        c[i] = (float*) _mm_malloc (size*sizeof(float),16);

```

```

        for (int j=0;j<size;j++)
            c[i][j] = a[i][j]+b[i][j];
    }
    return c;
}

__m128 ** add_mat_sse (__m128 ** a, __m128 ** b, int size) {
    int s = size/4;
    __m128 ** c = (__m128**) _mm_malloc (size*sizeof(__m128*),16);
    for (int i=0;i<size;i++) {
        c[i] = (__m128*) _mm_malloc (s*sizeof(__m128),16);
        for (int j=0;j<s;j++)
            c[i][j] = _mm_add_ps(a[i][j],b[i][j]);
    }
    return c;
}

float * add_vec (float * a, float * b, int size) {
    float * c = (float*) _mm_malloc (size*sizeof(float),16);
    for (int i=0;i<size;i++)
        c[i] = a[i]+b[i];
    return c;
}

__m128 * add_vec_sse (__m128 * a, __m128 * b, int size) {
    int s = size/4;
    __m128 * c = (__m128*) _mm_malloc (s*sizeof(__m128),16);
    for (int i=0;i<s;i++)
        c[i] = _mm_add_ps(a[i],b[i]);
    return c;
}

int main() {
    freopen("output.txt","w",stdout);
    srand(time(NULL));
    int size=8,min_number=-9,max_number=9;
    float ** a, ** b, ** c, * d, * e, * f, * g, h;
    __m128 ** as, ** bs, ** cs, * ds, * es, * fs, * gs, hs;

    printf("-- Multiplying 2 matrices\n");
    a = create_matrix(size,min_number,max_number);
    b = create_matrix(size,min_number,max_number);
    c = mul_mat(a,b,size);
    print_matrix(stdout,a,size,"A");
    print_matrix(stdout,b,size,"B");
    print_matrix(stdout,c,size,"C");
    as=(__m128 **)a;bs=(__m128 **)b;
    cs=mul_mat_sse(as,bs,size);
    print_matrix_sse(stdout,cs,size,"C");
    free_m(a,size);free_m(b,size);free_m(c,size);free_m_sse(cs,size);

    printf("\n-- Multiplying matrix on 4\n");
    a = create_matrix(size,min_number,max_number);
    h = 4;
    c = mul_num_mat(h,a,size);
    print_matrix(stdout,a,size,"A");
    print_matrix(stdout,c,size,"C");
    as=(__m128 **)a;

```

```

hs=_mm_set1_ps(h);
cs = mul_num_mat_sse(hs,as,size);
print_matrix_sse(stdout,cs,size,"C");
free_m(a,size);free_m(c,size);free_m_sse(cs,size);

printf("\n-- Multiplying matrix on vector\n");
a = create_matrix(size,min_number,max_number);
d = create_vector(size,min_number,max_number);
e = mul_mat_vec(a,d,size);
print_matrix(stdout,a,size,"A");
print_vector(stdout,d,size,"D");
print_vector(stdout,e,size,"E");
as((__m128 **))a;ds((__m128 *)d);
es = mul_mat_vec_sse(as,ds,size);
print_vector_sse(stdout,es,size,"E");
free_m(a,size);_mm_free(d);_mm_free(e);_mm_free(es);

printf("\n-- Adding 2 matrices\n");
a = create_matrix(size,min_number,max_number);
b = create_matrix(size,min_number,max_number);
c = add_mat(a,b,size);
print_matrix(stdout,a,size,"A");
print_matrix(stdout,b,size,"B");
print_matrix(stdout,c,size,"C");
as((__m128 **))a;bs((__m128 **))b;
cs = add_mat_sse(as,bs,size);
print_matrix_sse(stdout,cs,size,"C");
free_m(a,size);free_m(b,size);free_m(c,size);free_m_sse(cs,size);

printf("\n-- Adding 2 vectors\n");
d = create_vector(size,min_number,max_number);
e = create_vector(size,min_number,max_number);
f = add_vec(d,e,size);
print_vector(stdout,d,size,"D");
print_vector(stdout,e,size,"E");
print_vector(stdout,f,size,"F");
ds((__m128 *)d);es((__m128 *)e);
fs = add_vec_sse(ds,es,size);
print_vector_sse(stdout,fs,size,"F");
_mm_free(d);_mm_free(e);_mm_free(f);_mm_free(fs);

printf("\n-- Calculating formula\n");
printf("G = H*A*D+E, where H - number, D and E - vectors, A - matrix\n\n");
a = create_matrix(size,min_number,max_number);
d = create_vector(size,min_number,max_number);
e = create_vector(size,min_number,max_number);
h = rand_f(min_number,min_number);
printf("Number H is %i\n\n",(int)h);
print_matrix(stdout,a,size,"A");
print_vector(stdout,d,size,"D");
print_vector(stdout,e,size,"E");
b = mul_num_mat(h,a,size);
f = mul_mat_vec(b,d,size);
g = add_vec(f,e,size);
print_vector(stdout,g,size,"G");
as((__m128 **))a;ds((__m128 *)d);es((__m128 *)e);
hs=_mm_set1_ps(h);

```

```

bs = mul_num_mat_sse(hs,as,size);
fs = mul_mat_vec_sse(bs,ds,size);
gs = add_vec_sse(fs,es,size);
print_vector_sse(stdout,gs,size,"G");
free_m(a,size);_mm_free(d);_mm_free(e);
free_m(b,size);_mm_free(f);_mm_free(g);
free_m_sse(bs,size);_mm_free(fs);_mm_free(gs);

printf("\n-- Collecting statistics\n");
printf("- Multiplying 2 matrices\n");
clock_t t1,t2;
double dur[2];
FILE * file = fopen("1.csv","w");
for (size=100;size<=1000;size+=100) {
    a = create_matrix(size,min_number,max_number);
    b = create_matrix(size,min_number,max_number);
    t1=clock();
    c = mul_mat(a,b,size);
    t2=clock();
    dur[0] = 1.0*(t2-t1)/CLOCKS_PER_SEC;
    as=__m128 **a;bs=__m128 **b;
    t1=clock();
    cs=mul_mat_sse(as,bs,size);
    t2=clock();
    dur[1] = 1.0*(t2-t1)/CLOCKS_PER_SEC;
    free_m(a,size);free_m(b,size);free_m(c,size);
    free_m_sse(cs,size);
    printf("%i,%lf,%lf\n",size,dur[0],dur[1]);
    fprintf(file,"%i,%lf,%lf\n",size,dur[0],dur[1]);
}
fclose(file);

file = fopen("2.csv","w");
printf("\n- Calculating formula\n");
for (size=100;size<=4000;size+=100) {
    a = create_matrix(size,min_number,max_number);
    d = create_vector(size,min_number,max_number);
    e = create_vector(size,min_number,max_number);
    h = rand_f(min_number,min_number);
    t1=clock();
    b = mul_num_mat(h,a,size);
    f = mul_mat_vec(b,d,size);
    g = add_vec(f,e,size);
    t2=clock();
    dur[0] = 1.0*(t2-t1)/CLOCKS_PER_SEC;
    as=__m128 **a;ds=__m128 *d;es=__m128 *e;
    hs=_mm_set1_ps(h);
    t1=clock();
    bs = mul_num_mat_sse(hs,as,size);
    fs = mul_mat_vec_sse(bs,ds,size);
    gs = add_vec_sse(fs,es,size);
    t2=clock();
    dur[1] = 1.0*(t2-t1)/CLOCKS_PER_SEC;
    free_m(a,size);_mm_free(d);_mm_free(e);
    free_m(b,size);_mm_free(f);_mm_free(g);
    free_m_sse(bs,size);_mm_free(fs);_mm_free(gs);
    printf("%i,%lf,%lf\n",size,dur[0],dur[1]);
    fprintf(file,"%i,%lf,%lf\n",size,dur[0],dur[1]);
}

```

```

    }
    fclose(file);
    return 0;
}

```

## 7. Приклад

-- Multiplying 2 matrices

Matrix A:

5	-9	-3	5	5	7	-6	0
-3	6	-3	-3	-4	6	7	8
-7	2	-5	1	4	-9	1	-4
-9	-8	3	2	-8	-8	5	6
8	-8	-2	4	9	1	-9	-7
-6	-3	-1	-1	9	-7	-2	-8
1	-1	-1	-8	-4	7	-3	-6
8	7	5	6	8	-2	2	3

Matrix B:

-1	7	-6	-5	-5	-6	-3	-1
-3	2	4	-4	1	8	-6	-7
-3	3	-6	3	-3	-2	6	-5
-5	-2	-8	9	2	0	2	-8
-6	6	-7	-1	9	0	4	-3
8	-2	-1	0	-3	3	-1	0
-7	2	9	-1	9	2	0	-6
7	-2	9	-1	8	2	0	8

Matrix C:

74	2	-184	48	-45	-87	44	54
88	-50	241	-56	97	120	-73	34
-120	-10	26	20	94	35	6	-40
5	-108	151	101	77	-18	73	76
-30	74	-308	29	-93	-137	55	-3
-129	31	-108	31	48	-27	71	-39
104	-14	0	-63	-151	-9	-42	57
-131	123	-107	-12	84	2	10	-142

Matrix C with SSE:

74	2	-184	48	-45	-87	44	54
88	-50	241	-56	97	120	-73	34
-120	-10	26	20	94	35	6	-40
5	-108	151	101	77	-18	73	76
-30	74	-308	29	-93	-137	55	-3
-129	31	-108	31	48	-27	71	-39
104	-14	0	-63	-151	-9	-42	57
-131	123	-107	-12	84	2	10	-142

-- Multiplying matrix on 4

Matrix A:

-5	0	-3	-9	9	7	-3	4
-4	3	-9	-1	-4	-1	8	-5
-8	5	0	7	-3	9	8	-6
-2	7	0	-7	-4	6	-9	0
-4	-5	1	-8	-8	4	6	-6
-3	3	2	-1	-7	-2	3	-9
-7	-7	4	-4	9	2	-3	-6
-3	6	-4	2	2	-7	9	8

Matrix C:

-20	0	-12	-36	36	28	-12	16
-16	12	-36	-4	-16	-4	32	-20
-32	20	0	28	-12	36	32	-24
-8	28	0	-28	-16	24	-36	0
-16	-20	4	-32	-32	16	24	-24
-12	12	8	-4	-28	-8	12	-36
-28	-28	16	-16	36	8	-12	-24
-12	24	-16	8	8	-28	36	32

Matrix C with SSE:

-20	0	-12	-36	36	28	-12	16
-16	12	-36	-4	-16	-4	32	-20
-32	20	0	28	-12	36	32	-24
-8	28	0	-28	-16	24	-36	0
-16	-20	4	-32	-32	16	24	-24
-12	12	8	-4	-28	-8	12	-36
-28	-28	16	-16	36	8	-12	-24
-12	24	-16	8	8	-28	36	32

-- Multiplying matrix on vector

Matrix A:

-3	0	9	-4	-9	2	-1	-2
-4	8	6	-5	-4	-3	-8	-5
-4	5	-3	-5	-5	3	-2	1
-1	0	9	8	0	8	3	6
-2	2	8	-1	-8	-2	3	-6
-4	-4	-2	-2	2	-4	-1	8
-9	5	9	-5	-4	-5	2	1
1	2	-1	-9	0	-8	4	5

Vector D:

1	2	-9	-1	9	3	2	-8
---	---	----	----	---	---	---	----

Vector E:

-26	36	-315	23	-252	-27	0	-64
-----	----	------	----	------	-----	---	-----

Vector E with SSE:

-26	36	-315	23	-252	-27	0	-64
-----	----	------	----	------	-----	---	-----

-- Adding 2 matrices

Matrix A:

8	6	-4	-2	-8	4	-7	8
-3	-8	-9	2	-7	2	-7	4
-9	-1	4	1	-3	-2	-7	4
-9	-7	-7	6	2	1	8	-3
-3	-6	4	-5	5	3	-9	-8
5	-9	0	7	-1	-1	-2	0
7	-7	7	-5	-3	9	8	-6
8	8	-9	0	-1	5	3	3

Matrix B:

5	-2	7	-3	-9	4	-2	2
4	4	-3	-6	-6	4	0	7
3	-6	-8	7	-7	7	-9	7
2	7	-6	-9	2	6	3	-5

-9	-2	1	8	-8	6	0	3
0	3	3	0	-5	9	-2	4
-7	-4	1	-8	-7	-1	8	1
-4	-8	2	5	7	-8	9	8

Matrix C:

13	4	3	-5	-17	8	-9	10
1	-4	-12	-4	-13	6	-7	11
-6	-7	-4	8	-10	5	-16	11
-7	0	-13	-3	4	7	11	-8
-12	-8	5	3	-3	9	-9	-5
5	-6	3	7	-6	8	-4	4
0	-11	8	-13	-10	8	16	-5
4	0	-7	5	6	-3	12	11

Matrix C with SSE:

13	4	3	-5	-17	8	-9	10
1	-4	-12	-4	-13	6	-7	11
-6	-7	-4	8	-10	5	-16	11
-7	0	-13	-3	4	7	11	-8
-12	-8	5	3	-3	9	-9	-5
5	-6	3	7	-6	8	-4	4
0	-11	8	-13	-10	8	16	-5
4	0	-7	5	6	-3	12	11

-- Adding 2 vectors

Vector D:

-4	-3	3	-5	9	-7	4	-3
----	----	---	----	---	----	---	----

Vector E:

3	-3	6	7	-7	-6	-2	-5
---	----	---	---	----	----	----	----

Vector F:

-1	-6	9	2	2	-13	2	-8
----	----	---	---	---	-----	---	----

Vector F with SSE:

-1	-6	9	2	2	-13	2	-8
----	----	---	---	---	-----	---	----

-- Calculating formula

$G = H \cdot A \cdot D + E$ , where H - number, D and E - vectors, A - matrix

Number H is -9

Matrix A:

-3	9	-4	-4	-2	-6	7	0
-5	-4	-5	8	-3	-9	3	0
-5	-4	4	-6	-1	-5	0	8
-2	-6	2	0	-6	9	4	0
5	9	6	-7	9	3	0	-6
8	1	8	-8	2	2	1	3
7	-8	6	-7	-7	-6	-9	0
-3	-1	9	0	-1	3	-9	-6

Vector D:

-1	3	-6	-2	-7	0	-2	7
----	---	----	----	----	---	----	---



Vector E:  
-9 -3 8 2 8 -3 -5 5

Vector G:  
9 105 1412 -430 -559 -3 -59 68

Vector G with SSE:  
9 105 1412 -430 -559 -3 -59 68

```
-- Collecting statistics
- Multiplying 2 matrices
100,0.011975,0.006474
200,0.082635,0.030530
300,0.284289,0.108448
400,0.788482,0.281642
500,1.571325,0.561944
600,2.836564,1.074336
700,4.841930,1.740416
800,8.895281,3.985972
900,12.055253,4.898085
1000,19.605669,9.738386
```

```
- Calculating formula
100,0.000125,0.000048
200,0.000478,0.000186
300,0.001148,0.000434
400,0.002104,0.000876
500,0.003523,0.001469
600,0.005520,0.002327
700,0.007814,0.002868
800,0.011509,0.005718
900,0.022671,0.010001
1000,0.021528,0.013402
1100,0.026650,0.010397
1200,0.035869,0.017596
1300,0.036739,0.018358
1400,0.044098,0.025250
1500,0.050716,0.021340
1600,0.060784,0.031195
1700,0.072656,0.033197
1800,0.081367,0.042162
1900,0.090420,0.042685
2000,0.095099,0.052310
2100,0.140222,0.045190
2200,0.111707,0.058894
2300,0.117900,0.058572
2400,0.129483,0.070316
2500,0.136429,0.060709
2600,0.156198,0.082704
2700,0.155931,0.066116
2800,0.178979,0.093705
2900,0.187897,0.080826
3000,0.209602,0.107083
3100,0.216417,0.089779
3200,0.239477,0.127520
3300,0.249023,0.107179
3400,0.278025,0.138099
```

3500,0.281982,0.115805  
3600,0.312394,0.155651  
3700,0.321608,0.136493  
3800,0.353174,0.174369  
3900,0.356118,0.142215  
4000,0.391971,0.191166