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 cpuid 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 поток матрицю
   2. Аргументи:
      1. output – поток
      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 поток вектор
   2. Аргументи:

    1. output – поток
    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)

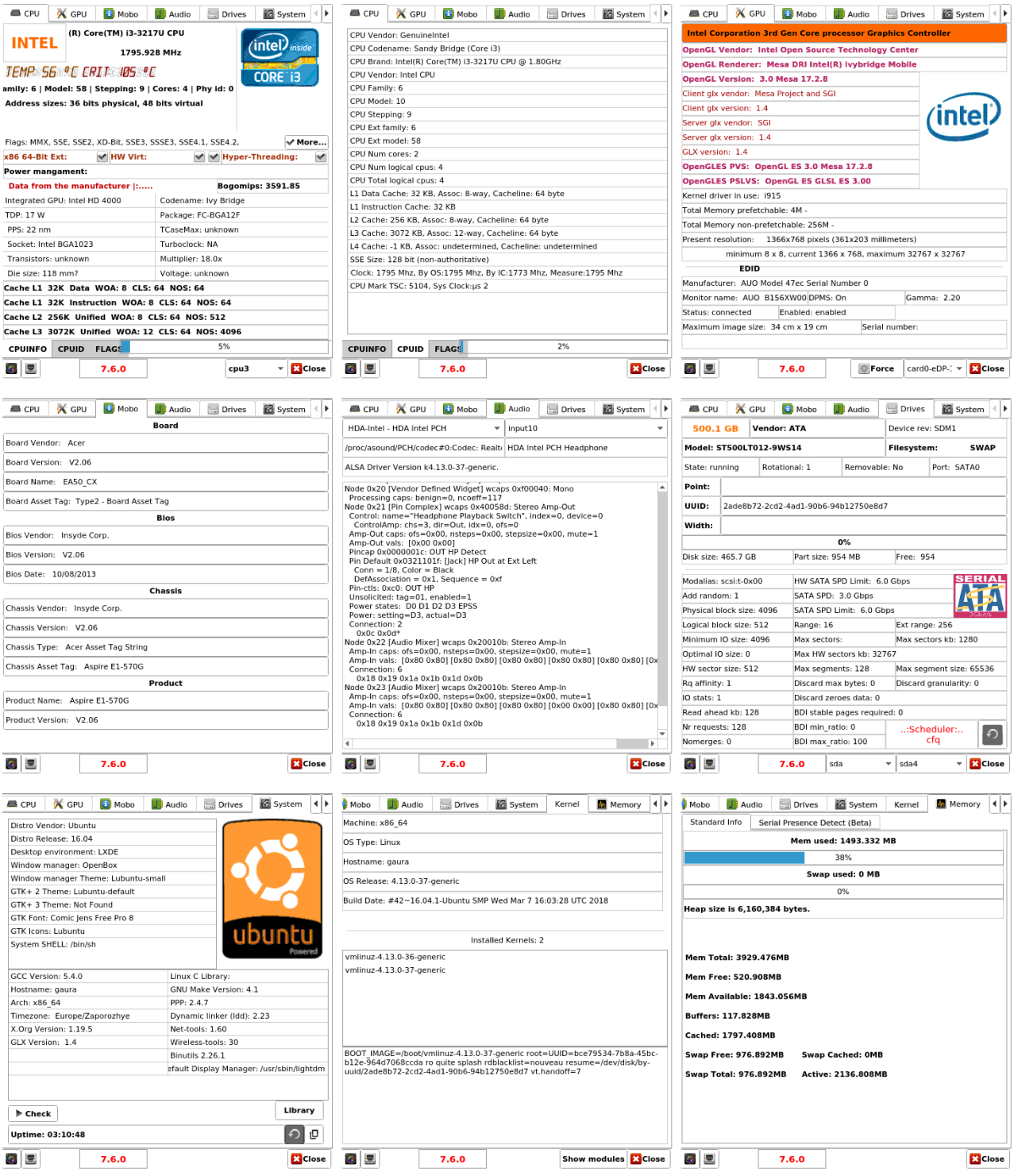
__m128 * add_vec_sse (__m128 * a, __m128 * b, int size)

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.

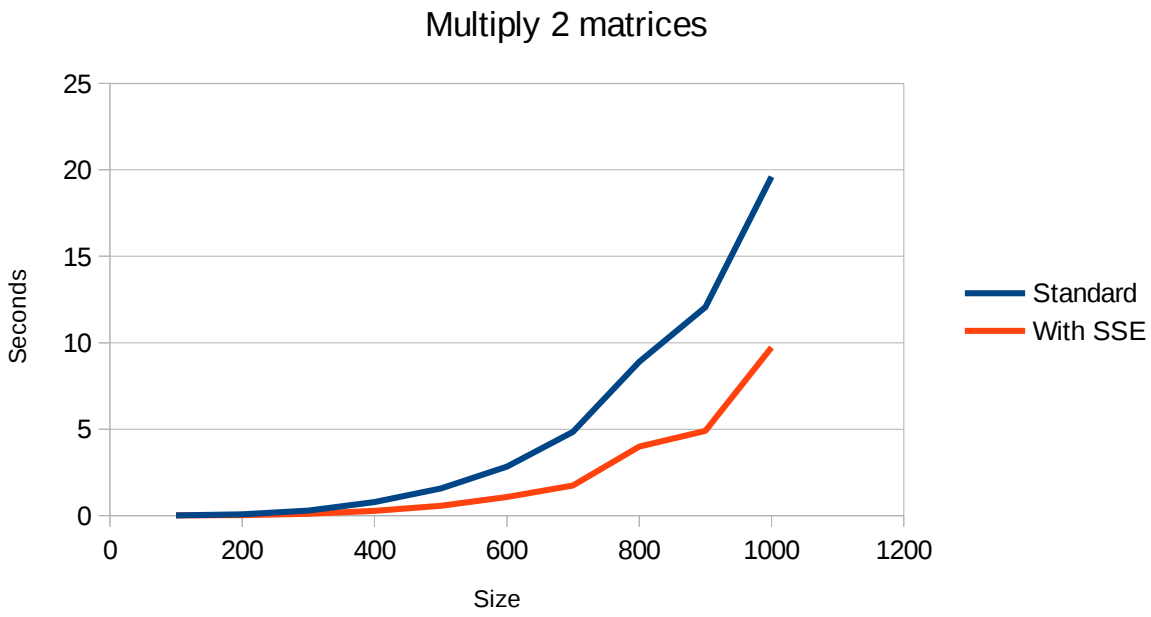
4. Інформація про комп'ютер:

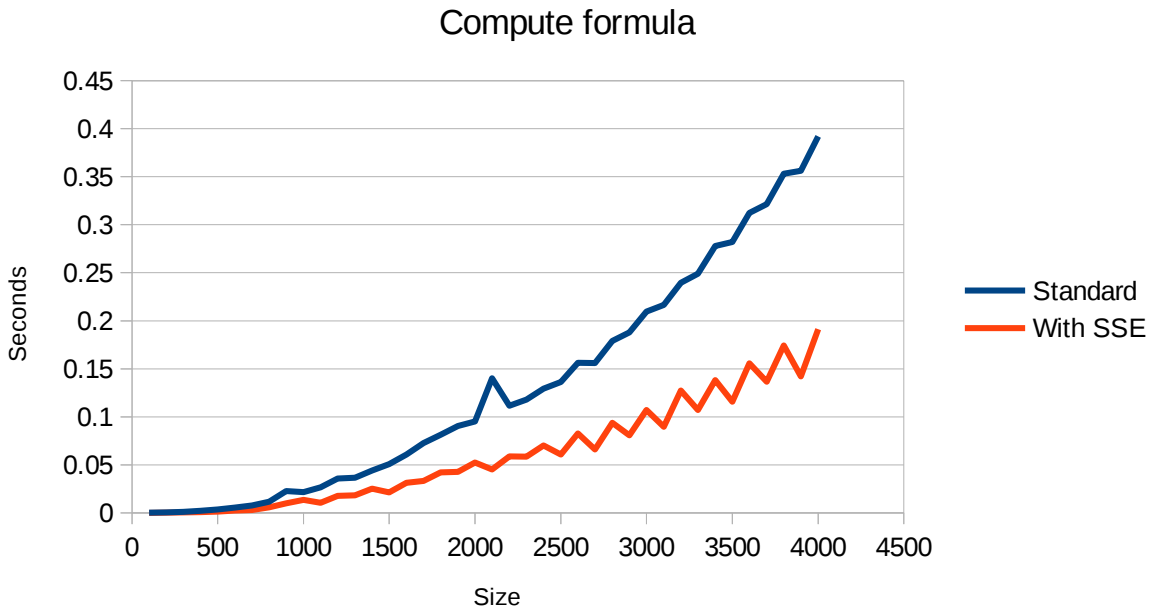


5. Таблиці та графіки:

Multiply 2 matrices		
Size	Standard	With SSE
100	0.011975	0.006474
200	0.082635	0.03053
300	0.284289	0.108448
400	0.788482	0.281642
500	1.571325	0.561944
600	2.836564	1.074336
700	4.84193	1.740416
800	8.895281	3.985972
900	12.055253	4.898085
1000	19.605669	9.738386

Compute formula		
Size	Standard	With SSE
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	0.00552	0.002327
700	0.007814	0.002868
800	0.011509	0.005718
900	0.022671	0.010001
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





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*A*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
```