УДК 004.522

Prof. **Myasishchev A.A.** (KhNU)
Prof. **Lienkov S.V.** (VIKNU)
PhD. **Dzhulii V.M.** (KhNU)
PhD. **Muliar I.V.** (KhNU)

USING GPU NVIDIA FOR LINEAR ALGEBRA PROLEMS

Research goals and objectives: the purpose of the article is to study the feasibility of graphics processors using in solving linear equations systems and calculating matrix multiplication as compared with conventional multi-core processors. The peculiarities of the MAGMA and CUBLAS libraries use for various graphics processors are considered. A performance comparison is made between the Tesla C2075 and GeForce GTX 480 GPUs and a six-core AMD processor.

Subject of research: the software is developed basing on the MAGMA and CUBLAS libraries for the purpose of the NVIDIA Tesla C2075 and GeForce GTX 480 GPUs performance study for linear equation systems solving and matrix multiplication calculating.

Research methods used: libraries were used to parallelize the linear algebra problems solution. For GPUs, these are MAGMA and CUBLAS, for multi-core processors, the ScaLAPACK and ATLAS libraries. To study the operational speed there are used methods and algorithms of computational procedures parallelization similar to these libraries. A software module has been developed for linear equations systems solving and matrix multiplication calculating by parallel systems.

Results of the research: it has been determined that for double-precision numbers the GPU GeForce GTX 480 and the GPU Tesla C2075 performance is approximately 3.5 and 6.3 times higher than that of the AMD CPU. And the GPU GeForce GTX 480 performance is 1.3 times higher than the GPU Tesla C2075 performance for single precision numbers. To achieve maximum performance of the NVIDIA CUDA GPU, you need to use the MAGMA or CUBLAS libraries, which accelerate the calculations by about 6.4 times as compared to the traditional programming method. It has been determined that in equations systems solving on a 6-core CPU, it is possible to achieve a maximum acceleration of 3.24 times as compared to calculations on the 1st core using the ScaLAPACK and ATLAS libraries instead of 6-fold theoretical acceleration. Therefore, it is impossible to efficiently use processors with a large number of cores with considered libraries. It is demonstrated that the advantage of the GPU over the CPU increases with the number of equations.

Key words. GPU Tesla C2075, MAGMA, CUBLAS, CUDA, graphics processor, NVIDIA Tesla V100, SMP system, MPI, ScaLAPACK, ATLAS.

Introduction. A large amount of computational mathematical problems are those solving linear algebraic equations systems. Many managerial and economic, technological problems are either constructed as linear algebraic equations, or are reduced to them. The widely used finite element method, which is used for solving almost all problems that are reduced to systems of partial differential equations, is reduced to solving linear equations systems as well.

Matrix methods are very popular in economic practice. They are used for statistical calculations, labor rationing, reduction of workflow, organization of internal production cost accounting, as well as for economic analysis. Solving linear equations systems is also reduced to matrix transformations, thus the development of efficient hardware and software is an important task.

Such tasks require high performance computing systems. Nowadays, computers with multi-core processors are widely used; they are in fact a parallel computing system with shared memory (SMP-system). With the occurrence of G80, the NVIDIA's chip of the eighth generation (2007), there has emerged CUDA, a software and hardware architecture, which allows performing calculations using NVIDIA GPUs. This technology is, just as in the case of the processor, a parallel computing system, which includes hundreds of processor cores. The problem is to parallelize the matrix multiplication and solve linear equation systems to get the maximum load of the multi-core CPU and GPU processor cores, and then to compare how much the GPU performance differs from that of the CPU for solving such problems with single and double precision numbers.

The purpose and objectives of the paper to study the graphics processors feasibility in solving, in particular, linear equations systems and matrix multiplication as compared to conventional multi-core processors. Peculiarities of the use and installation problems of the MAGMA library are considered.

To achieve the goal, two computing systems were used:

- the first one has an AMD Phenom II X6 1090T processor (CPU) with a 3.2 GHz frequency and a NVIDIA's Tesla C2075 graphics processor (GPU),

- the second one also has an AMD Phenom II X6 1090T processor (CPU) with a 3.2 GHz frequency, but a NVIDIA's GeForce GTX 480 GPU.

Analysis of recent research and publications and problem statement. There are numerous works devoted to the study of parallel systems performance, basing both on cluster systems, and multi-core processors [1-7]. For example, [3,4,8] present a number of tasks solutions using NVIDIA GPU. However, they didn't provide a detailed analysis of NVIDIA GPU's potential for both simple programming methods and programming methods based on cuBLAS (CUDA Basic Linear Algebra Subroutines), and MAGMA (Matrix Algebra on GPU and Multicore Architectures) program libraries [6,9] for solving linear equations systems and matrix multiplication with single and double precision numbers.

Currently, NVIDIA has introduced an accelerator Tesla V100 based on the Volta architecture [10]. This device has a high level of computing power, amounting to about 15 TFLOPS in single precision and 7.5 TFLOPS in double precision operations, which allows using it in modern super-computer systems [11]. NVIDIA also introduced a new family of GeForce RTX 20 Series GPUs [12] based on a new Turing architecture. Its computational capabilities for single-precision operations are about 12 teraflops and 0.37 teraflops for double precision ones. However, at present, budget GPUs based on NVIDIA Tesla C2075 and NVIDIA GeForce GTX 480 video cards, which are installed on many workstations used for high-performance computing, are still important. Therefore, we will focus on computational experiments for these GPUs, and a computer with an AMD Phenom II X6 1090T processor (CPU, 3.2 GHz) will be considered as a multi-core processor with SMP architecture.

Research methods to determine the considered computing systems performance. To solve the problems, a computational experiment was carried out to determine performance in GFLOP/s and calculation time on the two above mentioned computational systems for the matrix multiplication problems and linear equation systems solving. We used test examples from the ScaLAPACK, ATLAS, MAGMA, CUBLAS libraries [6]. We also used our own programs based on the considered libraries and traditional programming methods.

The results of the computing systems performance assessing. As indicated, two systems are considered for performing computational experiments. Moreover, in these systems, GPUs will act as computational accelerators for linear equations systems solving and matrix multiplication. In the first case, the calculation will be performed taking into account parallelization across the 6 cores of the processor using the MPI, ScaLAPACK and ATLAS libraries [1,2,7]. In the second and third cases parallelization is across the cores of the Tesla C2075 GPU and GeForce GTX 480 using the CUDA technology [3]. Computing systems are working on Linux Ubuntu. They have compilers Fortran F77, gfortran with the above mentioned libraries for a 6-core processor. For programming on the Tesla C2075 GPU and GeForce GTX 480, the NVIDIA's video driver and the CUDA Toolkit software are installed from http://developer.nvidia.com/cuda-toolkit-archive. The software installation sequence for the GPU is presented in the source [13]. Installing the MPI, ScaLAPACK, ATLAS libraries on Linux Ubuntu OS is presented in [6] for a system with a 4-core processor CORE 2 QUAD PENTIUM Q6600 2.4GHZ, therefore it is not considered here. Let us take a closer look at the MAGMA library installation and solution of arising problems.

The installation is similar for 32-bit and 64-bit Linux Ubuntu operating systems. For the Tesla C2075 GPU, it is advisable to use a 64-bit system, since the amount of global memory here is 6 GB. Before compiling MAGMA, you need to install the lapack library:

```
sudo apt-get install --yes --force-yes liblapack-dev
```

After copying the source magma_1.0.0.tar.gz from the website
http://www.cs.utk.edu/~tomov/magma_1.0.0.tar.gz unarchiving is performed

```
$gzip –d magma_1.0.0.tag.gz
$tar xf magma_1.0.0.tar
$cd magma_1.0.0
```

Using the nano editor, create the make.inc file with the following contents:

```
GPU_TARGET = 1
CC        = gcc
NVCC      = nvcc
FORT      = gfortran
ARCH      = ar
ARCHFLAGS = cr
RANLIB    = ranlib
OPTS      = -O3 -DADD_
FOPTS     = -O3 -DADD_ -x f95-cpp-input
NVOPTS    = --compiler-options -fno-strict-aliasing -DUNIX -O3 -DADD_
LDOPTS    = -fPIC -Xlinker -zmuldefs
LIB       = -lcblas  -llapack -lpthread -lcublas -lcudart  -lm
CUDADIR   = /usr/local/cuda
LIBDIR    =  -L/usr/local/cuda/lib -L/usr/lib
INC       = -I$(CUDADIR)/include
LIBMAGMA     = ../lib/libmagma.a
LIBMAGMABLAS = ../lib/libmagmablas.a
```

Here you need to pay attention to the value of the GPU_TARGET parameter. If you equate it to zero, the library is built for the GPU Tesla family, if you equate it to one, then this corresponds to Fermi family. GPU NVIDIA GeForce GTX 480 corresponds to Fermi family. For the Tesla C2075 GPU, let us build the Magma libraries with GPU_TARGET = 1 and GPU_TARGET = 0 and compare the speed of calculations.

Run the compile command make all. In this case, the libraries libmagma, libmagmablas will be created and the test cases will be compiled. However, there are problems.

1. For 32-bit system.

gfortran -O3 -DADD_ -x f95-cpp-input -Dmagma_devptr_t="integer(kind=../control/sizeptr.c: In function ‘main’: ../control/sizeptr.c:6: warning: format ‘%lu’ expects type …

Here you need to run the control / sizeptr.c program and bring it to the form

```
#include <stdlib.h>
#include <stdio.h>
int main (){  printf("%d", sizeof(void *));  return EXIT_SUCCESS;}
```

That means you need to change format % lu to format % d. The gfortran compiler does not issue errors FOR the 64-bit system.

2. When compiling test cases.

First problem:

```
../lib/libmagma.a(zlatrd.o): In function `magma_zlatrd':
zlatrd.cpp:(.text+0x332): undefined reference to `zdotc'
zlatrd.cpp:(.text+0xbe2): undefined reference to `zdotc'
```

You must run the src / zlatrd.cpp program and make changes.

```
   cblas_zdotc_sub(i_n, W(i +1, i), ione, A(i +1, i), ione, &value);
  //   blasf77_zdotc(&value, &i_n, W(i+1,i), &ione, A(i+1, i), &ione);
```

and

```
  cblas_zdotc_sub(i, W(0, iw), ione, A(0, i), ione, &value);
  //      blasf77_zdotc(&value, &i, W(0, iw), &ione, A(0, i), &ione);
```

Second problem:

```
../lib/libmagma.a(clatrd.o): In function `magma_clatrd':
clatrd.cpp:(.text+0x321): undefined reference to `cdotc'
clatrd.cpp:(.text+0xb30): undefined reference to `cdotc'
```

Similarly, you need to run the src / clatrd.cpp program and make changes

```
//          blasf77_cdotc(&value, &i, W(0, iw), &ione, A(0, i), &ione);
          cblas_cdotc_sub(i, W(0, iw), ione, A(0, i), ione, &value);
```

and

```
                cblas_cdotc_sub(i_n, W(i +1, i), ione, A(i +1, i), ione, &value);
```

// blasf77_cdotc(&value, &i_n, W(i+1,i), &ione, A(i+1, i), &ione);

It is evident that for the correct connection with the fortran libraries, the library libcblas is needed instead of libblasf77. The libcblas.so library is installed with liblapack installation and therefore it is described in the make.inc file in the LIB parameter.

The compilation of testing programs testing_sgemm.cpp and testing_dgemm.cpp of matrix multiplication with single and double precision numbers, is correspondingly performed by the command lines:

gcc -O3 -DADD_ -DGPUSHMEM=200 -I/usr/local/cuda/include -I../include -I../quark/include -c testing_dgemm.cpp -o testing_dgemm.o

gcc -O3 -DADD_ -DGPUSHMEM=200 -fPIC -Xlinker -zmuldefs -DGPUSHMEM=200 testing_dgemm.o -o testing_dgemm lin/liblapacktest.a -L../lib -lcuda -lmagma -lmagmablas -lmagma -L/usr/local/cuda/lib -L/usr/lib -lcblas -llapack -lpthread -lcublas -lcudart -lm

When running these programs, a calculation performance comparison is performed for cuBLAS and MAGMA libraries. Performance is fixed in GFlop/s. Before compiling these test programs, it is necessary to adjust the parameters istart = 1024 and iend = 10240, which specify the initial and final values of the matrices 1024 and 10240 dimensions. Too large values may not fit in the global memory of the GPU. Let us consider the results of the programs testing_sgemm.cpp and testing_dgemm.cpp for the GPU GeForce GTX 480:

```
./testing_sgemm
device 0: GeForce GTX 480, 1401.0 MHz clock, 1535.2 MB memory
Testing transA = N  transB = N
    M     N     K    MAGMA GFLop/s    CUBLAS GFlop/s      error
=================================================================
1024  1024  1024      670.04             649.38       0.000000e+00
1280  1280  1280      685.46             696.84       0.000000e+00
1600  1600  1600      756.35             698.86       0.000000e+00
2000  2000  2000      792.98             688.97       0.000000e+00
2500  2500  2500      763.24             779.94       0.000000e+00
3125  3125  3125      803.20             776.03       0.000000e+00
3906  3906  3906      812.69             776.12       0.000000e+00
4882  4882  4882      825.38             791.58       0.000000e+00...
6102  6102  6102      820.70             818.51       0.000000e+00
7627  7627  7627      824.92             826.61       0.000000e+00
9533  9533  9533      825.42             844.12       0.000000e+00

./testing_dgemm
device 0: GeForce GTX 480, 1401.0 MHz clock, 1535.2 MB memory
Testing transA = N  transB = N
    M     N     K    MAGMA GFLop/s    CUBLAS GFlop/s      error
=================================================================
1024  1024  1024      154.26             154.56       0.000000e+00
1280  1280  1280      161.57             161.80       0.000000e+00
1600  1600  1600      162.78             162.97       0.000000e+00
2000  2000  2000      155.17             160.45       0.000000e+00
2500  2500  2500      155.78             162.03       0.000000e+00
3125  3125  3125      162.14             161.00       0.000000e+00
3906  3906  3906      158.81             163.31       0.000000e+00
4882  4882  4882      161.21             163.45       0.000000e+00
6102  6102  6102      162.44             164.08       0.000000e+00
```

Comparing the results, we see that the larger the matrix size, the faster the CUBLAS library works. However, the difference in performance is not significant. The performance drops sharply

when moving from single to double precision numbers (approximately 5.1 times). Calculations for GPU GeForce GTX 480 were performed for the GPU_TARGET = 1 parameter (Fermi family).

Let us consider the similar calculation results for the GPU Tesla C2075 for the GPU_TARGET = 0 parameter (Tesla family) and GPU_TARGET=1 parameter (Fermi family).

1. Tesla family

```
./testing_sgemm
device 0: Tesla C2075, 1147.0 MHz clock, 5375.2 MB memory
Testing transA = N  transB = N
   M       N       K     MAGMA GFLop/s    CUBLAS GFlop/s       error
=========================================================
 1024  1024  1024      428.98                433.40        0.000000e+00
 1280  1280  1280      516.29                516.35        0.000000e+00
 1600  1600  1600      514.25                514.83        0.000000e+00
 2000  2000  2000      389.18                525.71        1.525879e-05
 2500  2500  2500      396.82                604.98        1.525879e-05
 3125  3125  3125      405.15                573.97        1.525879e-05
 3906  3906  3906      401.76                579.76        3.051758e-05
 4882  4882  4882      404.92                586.02        3.051758e-05
 6102  6102  6102      402.93                613.92        3.051758e-05
 9533  9533  9533      409.14                639.26        6.103516e-05

./testing_dgemm
device 0: Tesla C2075, 1147.0 MHz clock, 5375.2 MB memory
Testing transA = N  transB = N
   M       N       K     MAGMA GFLop/s    CUBLAS GFlop/s       error
=========================================================
 1024  1024  1024      169.47                280.13        0.000000e+00
 1280  1280  1280      172.29                290.61        0.000000e+00
 1600  1600  1600      173.52                294.22        0.000000e+00
 2000  2000  2000      162.38                287.59        2.842171e-14
 2500  2500  2500      163.30                285.84        2.842171e-14
 3125  3125  3125      165.76                278.25        2.842171e-14
 3906  3906  3906      163.75                291.70        5.684342e-14
 4882  4882  4882      164.99                291.50        5.684342e-14
 6102  6102  6102      162.18                293.82        5.684342e-14
 6102  6102  6102      162.18                293.82        5.684342e-14
 9533  9533  9533      166.24                293.68        1.136868e-13
```

In this case, the MAGMA library has significantly lower performance than the CUBLAS library for both single-precision numbers (about 1.6 times) and double-precision numbers (about 1.8 times)

2. Fermi family

```
./testing_sgemm
device 0: Tesla C2075, 1147.0 MHz clock, 5375.2 MB memory
Testing transA = N  transB = N
   M       N       K     MAGMA GFLop/s    CUBLAS GFlop/s       error
=========================================================
 1024  1024  1024      551.34                431.39        7.629395e-06
 1280  1280  1280      566.34                516.79        7.629395e-06
 1600  1600  1600      600.45                514.57        7.629395e-06
 2000  2000  2000      617.14                525.37        1.525879e-05
 2500  2500  2500      585.90                605.09        1.525879e-05
 3125  3125  3125      622.99                573.94        1.525879e-05
```

3906	3906	3906	629.77	579.80	3.051758e-05
4882	4882	4882	640.83	585.94	3.051758e-05
6102	6102	6102	637.69	613.76	3.051758e-05
9533	9533	9533	639.57	639.24	6.103516e-05

./testing_dgemm
device 0: Tesla C2075, 1147.0 MHz clock, 5375.2 MB memory
Testing transA = N transB = N

M	N	K	MAGMA GFLop/s	CUBLAS GFlop/s	error
1024	1024	1024	279.11	280.61	0.000000e+00
1280	1280	1280	290.00	290.87	0.000000e+00
1600	1600	1600	294.01	294.44	0.000000e+00
2000	2000	2000	281.17	287.59	2.842171e-14
2500	2500	2500	282.24	285.60	2.842171e-14
3125	3125	3125	295.79	278.13	2.842171e-14
3906	3906	3906	289.47	291.83	5.684342e-14
4882	4882	4882	293.89	291.59	5.684342e-14
6102	6102	6102	295.99	293.87	5.684342e-14
9533	9533	9533	300.77	293.83	1.136868e-13

In this case, the MAGMA library has a performance slightly higher than CUBLAS. Therefore, for the GPU Tesla C2075, the compilation of the MAGMA libraries with the GPU_TARGET = 1 parameter (the make.inc file) is necessary.

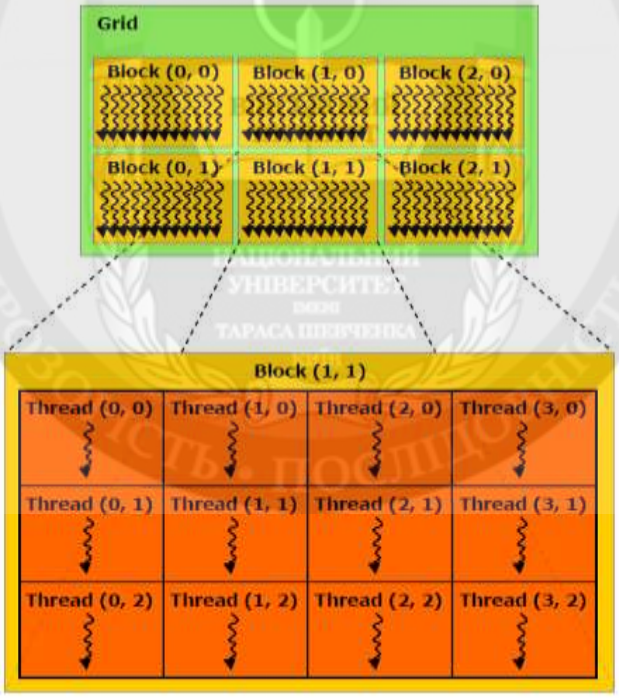

Figure 1 – The GPU program model

Let us analyze how much higher is the GPU's efficiency with NVIDIA's CUDA technology as compared to traditional way of writing parallel programs. To do this, we compare the efficiency of the matrix multiplication program using the global GPU memory (the first program), which is compiled according to the method described in [3, 4] with the program using the MAGMA and CUBLAS libraries (the second program).

To compile a program using global GPU memory, let us consider the GPU program model for the C compiler [4]. The top level of the core, the grid, consists of blocks. The blocks are either one-

dimensional or two-dimensional network of threads. This can be illustrated by Fig.1. The thread number in the block or the block number in the grid can be a variable of int or dim3 type.

The number of threads included in a block is determined by the blockDim built-in variable. The thread index inside the block is determined by the threadIdx variable, and the block index inside the grid is defined by the blockIdx variable. The threads in the block are directly performing of calculations. The thread is a 3-component vector, i.e. it can be identified using one-, two- and three-dimensional index of the thread, forming in turn a one-, two- and three-dimensional block of threads. There is a limit on the number of threads per block, which cannot exceed 1024 threads. The CUDA C extension allows you to call a function called the core so that it will execute in parallel N different CUDA threads. Such a function is declared with the __global__ specifier. Below is the first program compiled in accordance with the GPU's program model.

First program:

```
#include <stdio.h>
#define BLOCK 16 // Set the block size
// Multiplication function of two matrices
__global__ void mulMatr(float* a, float* b, float* c, int n)
{ // Get the id of the current thread.
  int i = threadIdx.y+blockIdx.y*blockDim.y; int j = threadIdx.x+blockIdx.x*blockDim.x;
  // Calculate the result.
     float sum=0.0f;
     for(int p = 0; p < n; p++){ sum+=  a[i*n + p] * b[p*n + j];}  c[i*n+j] = sum;}
__host__ int main()
{int N; int M; float mf=0.0f; printf ( "Input N->");  scanf ( "%d",&N);
 printf ( "Matrix = %dx%d elements\n", N,N );  M=N*N;
// Allocate memory for vectors
  float* a = new float[M]; float* b = new float[M]; float* c = new float[M];
// Initialize the values of the vectors
  for (int i = 0; i < N; i++){
   for (int j = 0; j < N; j++) { a[i*N+j] = 1.0f*((i+1)+2*(j+1)); b[i*N+j] = 1.0f/a[i*N+j];}}
// Video Memory Pointers
  float* deva;float* devb;float* devc;
// Allocate memory for vectors on the video card
  cudaMalloc((void**)&deva, sizeof(float) * M);cudaMalloc((void**)&devb, sizeof(float) * M);
  cudaMalloc((void**)&devc, sizeof(float) * M);
// create cuda event handle
cudaEvent_t start, stop;float gpuTime=0.0f;cudaEventCreate ( &start );cudaEventCreate ( &stop );
// Copy the data to the video card
  cudaMemcpy(deva, a, sizeof(float) * M, cudaMemcpyHostToDevice);
  cudaMemcpy(devb, b, sizeof(float) * M, cudaMemcpyHostToDevice);
// Execute the core function call
dim3 threads = dim3(BLOCK,BLOCK); // The number of threads in the block
dim3 blocks  = dim3(N/BLOCK,N/BLOCK); // The number of blocks in the grid
cudaEventRecord(start, 0); // connect the event to the core execution start
mulMatr<<<blocks, threads>>>(deva, devb, devc,N);
cudaEventRecord(stop, 0); // connect the event to the core execution ending. We get the result of
the calculation
 cudaMemcpy(c, devc, sizeof(float) * M, cudaMemcpyDeviceToHost);
cudaEventSynchronize(stop); // Wait for the core to execute, synchronizing on the stop event.
cudaEventElapsedTime (&gpuTime,start,stop);// Request time between start, stop
  // Calculation results
 gpuTime=gpuTime/1000;  mf=((2.0*N-1)*N*N)/(gpuTime*1000000.0);
 printf("time=%.4fsec\nspeed=%0.2fMFlops\n",gpuTime,mf);
 printf("i=%d\tj=%d\tC=%.5f\n",N/256,N/128,c[(N/256)*N+(N/128)]);
```

```
 printf("i=%d\tj=%d\tC=%.5f\n",3*N/4,5*N/16,c[(3*N/4)*N+(5*N/16)]);
 printf("i=%d\tj=%d\tC=%.5f\n",N-4,N-2,c[(N-4)*N+(N-2)]);
  // Releasing resources
  cudaEventDestroy(start); cudaEventDestroy(stop); cudaFree(deva);cudaFree(devb);cudaF-
ree(devc);
  delete[] a; a = 0;delete[] b; b = 0;delete[] c; c = 0; }
```

This program is compiled by the command:

```
nvcc -arch=sm_20 matrmul_g.cu -o matrmul_g
```

Here matrmul_g.cu is the name of the program source code.

The second program presents a simplified version of the testing_sgemm.cpp and testing_dgemm.cpp test cases, when the dimension of the matrices is a multiple of 16.

Second program:

```
#include <stdio.h>
#include <cuda.h>
#include <cublas.h>
#include "magma.h"
#include "magma_lapack.h"
int main ( int argc, char** argv )
{float time_seconds=0.0f; float mf=0.0f;
cudaEvent_t start, stop;cudaEventCreate ( &start );cudaEventCreate ( &stop );
int N=6144; printf ( "Input N->");if (scanf ( "%d",&N)==1);{printf ( "Matrix = %dx%d ele-
ments\n", N,N );}
int M=N*N; float *d_A, *d_B, *d_C;float* A = new float[M]; float* B = new float[M];
float* C = new float[M];
for (int j = 0; j < N; j++){for (int i = 0; i < N; i++){A[i+j*N] = 1.0*((i+1)+2*(j+1));B[i+j*N] =
1.0/A[i+j*N];}}
cublasInit(); cublasAlloc ( N * N, sizeof(float), (void**)&d_A);
cublasAlloc ( N * N, sizeof(float), (void**)&d_B);cublasAlloc ( N * N, sizeof(float),
(void**)&d_C);
cublasSetMatrix ( N, N, sizeof(float), (void *) A, N, (void *) d_A, N);
cublasSetMatrix ( N, N, sizeof(float), (void *) B, N, (void *) d_B, N);
cudaEventRecord(start, 0); magmablas_sgemm( 'n', 'n', N, N, N, 1.0f, d_A, N, d_B, N, 0.0f, d_C, N
);
cudaEventRecord(stop, 0); cublasGetMatrix ( N, N, sizeof(float), (void *) d_C, N, (void *) C, N );
 cudaEventElapsedTime (&time_seconds,start,stop);
 cublasFree (d_A);cublasFree (d_B); cublasFree (d_C); cublasShutdown();
 time_seconds=time_seconds/1000; mf=((2.0*N-1)*N*N)/(time_seconds*1000000.0);
 printf("time=%.4fsec\nspeed=%0.2fMFlops\n",time_seconds,mf);
 printf("i=%d\tj=%d\tC=%.5f\n",N/256,N/128,C[(N/256)+(N/128)*N]);
 printf("i=%d\tj=%d\tC=%.5f\n",3*N/4,5*N/16,C[(3*N/4)+(5*N/16)*N]);
 printf("i=%d\tj=%d\tC=%.5f\n",N-4,N-2,C[(N-4)+(N-2)*N]); }
```

These two programs are written for single-precision numbers. For double-precision numbers it is necessary to change the float to double data type. Also in the second program, the magmablas_sgemm function must be replaced by magmablas_dgemm one. The second program uses the library MAGMA. To use CUBLAS, instead of the magmablas_sgemm function, you must enter the cublasSgemm function in the program. The program is compiled by commands:

```
gcc -O3 -DADD_ -DGPUSHMEM=200 -I/usr/local/cuda/include -I../include -I../quark/in-
clude -c mb.cpp -o mb.o
gcc -O3 -DADD_ -DGPUSHMEM=200 -fPIC -Xlinker -zmuldefs -DGPUSHMEM=200 mb.o
-o mb -L../lib -lcuda -lmagma -lmagmablas -lmagma -L/usr/local/cuda/lib64 -L/usr/lib -lcblas -
llapack -lpthread -lcublas -lcudart -lm
```

Here mb.cpp is the name of the second program.

Table 1 presents the results of above mentioned programs performance estimates for the double-precision numbers for the Tesla C2075 GPUs and those of the programs presented in [14] using the ScaLAPACK and ATLAS libraries for the AMD Phenom II X6 1090T CPUs. In table 1, the numerator is the counting time in seconds, the denominator is the performance in Gigaflops per second. Performance was defined as the ratio of the floating-point operations number in the matrix product to the elapsed time. The number of operations in programs is determined by the expression: op = N * N (2 * N-1), where N is the number of rows or columns of a square matrix.

Table 1

Results of performance evaluations for the Tesla C2075 GPU and for AMD Phenom II X6 1090T CPU

Matrix NxN	AMD processor Phenom II X6 1090T (6 cores)	GPU Tesla C2075				
		Global memory	Tesla Compilation		Fermi Compilation	
			CUBLAS	MAGMA	CUBLAS	MAGMA
1024	0,062/34,6	0,045/47,3	0,008/284,4	0,013/170,2	0,008/284,8	0,008/282,4
2048	0,407/42,2	0,375/45,9	0,058/295,9	0,099/173,8	0,058/295,9	0,058/295,2
3072	1,234/47,0	1,233/47,0	0,194/299,3	0,333/174,1	0,194/299,4	0,194/298,9
4096	2,870/47,9	2,963/46,4	0,458/300,3	0,792/173,5	0,458/300,2	0,458/299,9
5120	5,687/47,2	5,715/47,0	0,893/300,6	1,536/174,7	0,893/300,5	0,894/300,2
6144	- / -	10,025/46,3	1,542/300,8	2,654/174,7	1,543/300,6	1,543/300,6

Comparison of the data in Table 1 with the results of test programs showed their close similarity. The performance of the processor is comparable with the performance of the program for the Tesla C2075 GPU, written using global memory [3] and about 6.3 times lower for the Tesla C2075 GPU for programs using the CUBLAS and MAGMA libraries.

Solving linear equation systems. Let us consider linear equation systems solving. In the test program on FORTRAN the calculation is made for double-precision numbers. Solution of the equations system for AMD CPU is divided into two stages:

- system matrix factorization (A) [15];
- the system solving using the results obtained at the first stage.

Each stage is performed by referring to the corresponding subroutines of the ScaLAPACK, BLACS, ATLAS libraries. The first stage uses the PDGETRF subroutine, the second uses the PDGETRS subroutine [7]. The program counts for each subroutine the time of its execution. The number of operations for the first (ops) and second stage (ops1) is calculated, and the system performance is calculated for LU factorization and the solution of the system as a whole.

Let us consider the procedure for solving the same problem on the GPU. To ensure maximum performance, we will also use ready-made libraries for solving equations systems using the LU factorization method. For this purpose, lapack-3.4.0 and MAGMA libraries (Matrix Algebra on GPU and Multicore Architectures) were installed on the computing system [9].

Below there is the ur_f1.f90 program, written on FORTRAN 90 for the GPU, for equations system solving using the LU factorization method:

```
program testing_dgetrf_gpu_f
use magma
external cublas_init, cublas_set_matrix, cublas_get_matrix
external cublas_shutdown, cublas_alloc
integer cublas_alloc
double precision, allocatable :: h_A(:), h_B(:), h_X(:)
magma_devptr_t              :: devptrA, devptrB
integer,  allocatable       :: ipiv(:)
integer                     :: i, n, info, stat, lda, size_of_elt, nrhs
real(kind=8)                :: ops,ops1, t, t1
integer                     :: tstart(2), tend(2), tstart1(2), tend1(2)
```

```
      call cublas_init()
      write (*,*) "Input n"
      read (*,*) n
      nrhs = 1
      lda  = n
      ldda = ((n+31)/32)*32
      size_of_elt = sizeof_double
      allocate(h_A(lda*n))
      allocate(h_B(lda*nrhs))
      allocate(h_X(lda*nrhs))
      allocate(ipiv(n))
      stat = cublas_alloc(ldda*n, size_of_elt, devPtrA)
      stat = cublas_alloc(ldda*nrhs, size_of_elt, devPtrB)
      do 10 i=1,n
      do 10 j=1,n
      h_A(i+(j-1)*n)=(sqrt(i/1.0d0))/11.0d0+(sqrt(j/1.0d0))/12.0d0
      if(i.eq.j) h_A(i+(j-1)*n)=10.0d0
10    continue
      do 11 i=1,n
      h_B(i)=(sqrt(i/1.0d0))/20.0d0 - 1.0d0
11    continue
      h_X(:) = h_B(:)
      call cublas_set_matrix(n, n, size_of_elt, h_A, lda, devptrA, ldda)
      call cublas_set_matrix(n, nrhs, size_of_elt, h_B, lda, devptrB, ldda)
      call magma_gettime_f(tstart)
      call magmaf_dgetrf_gpu(n, n, devptrA, ldda, ipiv, info)
      call magma_gettime_f(tend)
      call magma_gettimervalue_f(tstart, tend, t)
      write(*,*) "time LU =", t
      call magma_gettime_f(tstart1)
      call magmaf_dgetrs_gpu('n', n, nrhs, devptrA, ldda, ipiv, devptrB, ldda, info)
      call magma_gettime_f(tend1)
      call magma_gettimervalue_f(tstart1, tend1, t1)
      write(*,*) "time solve =", t1
      call cublas_get_matrix (n, nrhs, size_of_elt, devptrB, ldda, h_X, lda)
      write(*,*) "X(1)=",h_X(1),"X(",n,")=",h_X(n)
      ops = 2.0d0 * (dfloat(n))**3 / 3.0d0
      ops1 = 2.0d0 * (dfloat(n))**3 / 3.0d0 + 2.0d0*(dfloat(n))**2
      write(*,*)   ' Gflops LU  = ', ops /(t*1e6)
      write(*,*)   ' Gflops summa solve  = ', ops1 /(1e6*(t+t1))
      deallocate(h_A, h_X, h_B, ipiv)
      call cublas_free(devPtrA)
      call cublas_free(devPtrB)
      call cublas_shutdown()
      end
```

Here, the subroutine magmaf_dgetrf_gpu is used for LU factorization, and magmaf_dgetrs_gpu is used to solve the equations system [9,16]. The execution time and computational performance for them are also considered separately for LU factorization and the solution of the system as a whole.

Let us consider the results of computational experiments. Table 2 summarizes the results of calculations for the programs presented above for the AMD CPU and the Tesla C2075 GPU with double precision calculations. Two options are given for the processor: the calculation is performed by one core and 6 cores. The presented acceleration value does not reach the sixfold value anywhere. The results are presented as a fraction where the numerator is the counting time in seconds, and the

denominator is the performance in Gigaflops per second. Separate columns present calculations for LU factorization and system solutions. The "solution" column numerator gives the working time of the PDGETRS subroutine, i.e. return trace, and the denominator shows the total performance of system solving. It is ovious that the main complexity of the calculations is associated with LU-factorization. For a CPU, it is 26.491 / 0.121 = 219 times longer than the return trace time, and for a GPU it is 3.718 / 0.071 = 52 times longer for a 11008x11008 matrix. The ratio increases with the equations size. Thus, performance evaluation can only be performed by LU factorization.

Table 2

Calculation results for AMD CPU and Tesla C2075 GPU

Matrix NxN	CPU AMD Phenom II X6 1090T					GPU Tesla C2075	
	6 cores		1 core		Acceleration	LU-factor.	Solution
	LU-fact.	Solution	LU-fact.	Solution			
1920	0.31/15.4	0.01/14.9	0.60/7.89	0.01/7.77	1.92	0.076/61.7	0.010/54.4
3072	0.91/21.1	0.02/20.7	2.38/8.11	0.03/8.03	2.58	0.230/84.0	0.017/78.4
4992	2.62/31.7	0.03/31.3	8.51/9.75	0.06/9.68	3.24	0.518/160.0	0.029/151.7
7104	7.61/31.4	0.05/31.2	23.52/10.2	0.12/10.1	3.09	1.180/202.6	0.043/195.6
8064	10.88/32.1	0.07/31.9	33.92/10.3	0.15/10.3	3.10	1.684/207.6	0.050/201.7
9984	20.23/32.8	0.10/32.6	61.90/10.7	0.24/10.7	3.05	2.882/230.2	0.063/225.3
11008	26.49/33.6	0.12/33.4	82.28/10.8	0.28/10.8	3.09	3.718/239.2	0.071/234.7
25344	-	-	-	-	-	38.665/280.7	0.204/279.2

Table 3 shows the results of performance calculations of LU factorization in GFlop / s for the GPU Tesla C2075, GeForce GTX 480 and AMD Phenom II X6 1090T CPU for single (real) and double (double) precisions. It can be seen that for the equations size up to 3072, the cheaper GPU GeForce 480 overtakes the Tesla C2075 GPU. The Tesla C2075 has advantage for big equations. In general, with double-precision calculations for the 11008 equations size, the GPU Tesla C2075 overtakes the processor by 239.2 / 34.1 = 7.01 times.

Table 3

Results of productivity calculations for LU factorization in GFlop/s

Matrix NxN	GPU Tesla C2075		GPU GeForce 480		CPU AMD	
	real	double	real	double	real	double
1920	77.9	61.7	90.9	66.4	18.3	15.4
3072	**132.3**	**84.2**	**170.1**	**84.2**	**30.8**	**21.1**
4992	150.9	160.3	161.2	128.5	50.0	31.7
7104	221.4	203.3	234.3	143.1	60.9	32.1
8064	251.4	208.1	263.4	145.0	58.5	32.7
9984	317.3	230.2	329.7	150.9	71.6	33.3
11008	351.4	239.2	365.3	153.2	73.9	34.1

Table 4 shows the first argument calculations results of the equations system. Here Xr (1) corresponds to single precision, and Xd (1) corresponds to double precision. It is clear that the smaller the equations size, the less Xr (1) and Xd (1) differ. With a system size of 3000 equations, you can "believe" only the first three significant digits of the first argument, and with a size of 10,000 equations, the error becomes so large that the first significant digits become questionable. Thus, for systems with over 3000 equations, it is advisable to use only double-precision numbers in calculations.

Table 4

The results of the first argument calculations of the equations system

Matrix NxN	Argument Xr(1), Single precision.	Argument Xd(1), Double precision	Difference |Xr(1)-Xd(1)|
100x100	-9.904994E-02	-9.904992E-02	2.0E-06
1000x1000	7.36474E-03	7.36434E-03	4.0E-04
2000x2000	2.50738E-03	2.50710E-03	2.80E-04
3000x3000	**1.16325E-03**	**1.16273E-03**	**4.80E-04**
4000x4000	4.93012E-04	5.01715E-04	8.70E-02
5000x5000	4.92534E-04	5.03876E-04	1.13E-01
10000X10000	7.34304E-04	7.90892E-04	5.66E-01

Solving a system of 11008 equations with double precision numbers for Xd (1) showed the following results:

GPU Tesla C2075 Xd(1)=**9.93926624**47128E-004

CPU AMD Phenom II X6=**9.93926624**52686E-004

Here we can assume that the first nine significant digits are correct.

Conclusions

1. The installation process of the MAGMA libraries is not self-adjusting. Often it is necessary to make manual adjustments to the libraries when compiling and change the program texts. Therefore, the libraries' installation requires sufficient qualifications of a system programmer.

2. The MAGMA and CUBLAS libraries showed approximately the same matrix product performance for both single and double precision numbers. However, performance drops dramatically for the GeForce GTX 480 GPU when moving from single-precision to double-precision numbers (about 5.1 times). For the Tesla C2075 GPU, the performance drop does not exceed 2.1 times. Thus, Tesla C2075 GPU is more suitable for solving complex problems, requiring calculations with double precision. It also has a global memory of 6 GB (GeForce GTX 480 GPU has 1.5 GB).

3. The GPU GeForce GTX 480 and GPU Tesla C2075 performance for double-precision numbers is higher than that of the AMD Phenom II X6 1090T CPU by approximately 3.5 and 6.3 times respectively. And the performance of the GPU GeForce GTX 480 is 1.3 times higher than the performance of the GPU Tesla C2075 for single precision numbers. Therefore, for small tasks that require no more than 1.5 GB of memory and calculations with single precision, it is better to use GeForce GTX 480 GPU as an inexpensive and very efficient solution.

4. To achieve maximum performance of the NVIDIA CUDA GPU, it is necessary to use the MAGMA or CUBLAS libraries, which give an acceleration of the considered calculations about 6.4 times as compared with the use of global memory (the traditional programming method).

5. When solving equations systems, performance can be estimated only by LU factorization.

6. For systems with over 3000 equations, it is advisable to use only double-precision numbers in calculations.

7. When solving systems with up to 3000 equations, it is advisable to use a cheaper GPU GeForce 480 for computing with any precision. It works in this range of equations faster than AMD Phenom II X6 1090T CPU by 4 times for double precision numbers and 5.5 times for single precision numbers.

8. For a system of 11008 equations with double-precision GPU computations, it "excels" the CPU 7 times.

9. When solving systems of equations on a 6-core CPU, it is possible to achieve a maximum acceleration of 3.24 times as compared to calculations on the 1st core for the ScaLAPACK and ATLAS libraries. Sixfold acceleration cannot be obtained. Therefore, the effective use of processors with a large number of cores with the considered libraries is questionable.

10. The advantage of the GPU over the CPU increases with the number of equations. For example, the Tesla C2075 GPU for the 1920 equations in the system is 4 times faster than AMD CPU, and for 11008 equations it is even 7 times faster.

11. The considered methods for linear equations systems solving and matrix multiplication are valid for use in the single GPU computing system. However, it is not known how effective the parallelization of this task is on several GPUs installed in a multi-core system.

д.т.н., проф. Мясищев О.А., д.т.н., проф. Ленков С.В.,
к.т.н., доц. Джулий В.Н., к.т.н., доц. Муляр И.В.

ИСПОЛЬЗОВАНИЯ GPU NVIDIA ПРИ РЕШЕНИИ ЗАДАЧ ЛИНЕЙНОЙ АЛГЕБРЫ

В работе исследуется целесообразность применения графических процессоров при решении систем линейных уравнений и расчета матричного умножения по сравнению с обычными многоядерными процессорами. Рассматриваются особенности использования и проблемы установки библиотеки MAGMA. Для проведения вычислительных экспериментов рассмотрены две системы. В каждой из них установлен шестиядерный процессор (CPU) AMD. В первой системе использован графический процессор (GPU) Tesla C2075, во второй GeForce GTX 480 фирмы NVIDIA. GPU выполняют роль вычислительных ускорителей для решения систем линейных уравнений и матричного умножения. Причем в первом случае расчет выполняется с учетом распараллеливания по 6-ядрам процессора с использованием библиотек MPI, ScaLAPACK и ATLAS. Во втором и третьем случаях – распараллеливанием по ядрам GPU Tesla C2075 и GeForce GTX 480 с использованием технологии CUDA. Вычислительные системы работают под управлением операционной системы теками для 6-и ядерного процессора. Для программирования на GPU Tesla C2075 и GeForce GTX 480 инсталлированы видеодрайвер nvidia и программное обеспечение CUDA Toolkit. Установлено, что производительность GPU GeForce GTX 480 и GPU Tesla C2075 выше производительности тельность GPU GeForce GTX 480 в 1.3 раза выше производительности GPU Tesla C2075 для чисел с одинарной точностью. Показано, что для достижения максимальной производительности ускорение расчетов примерно в 6.4 раза по сравнению с традиционным способом программирования. Установлено, что при решении систем уравнений на 6-и ядерном CPU удается достичь максимального ускорения в 3.24 раза по сравнению с расчетами на 1-м ядре при использовании библиотек ScaLAPACK и ATLAS. Шестикратного ускорения получить не удается. Поэтому эффективное использование процессоров с большим количеством ядер с рассмотренными библиотеками стоит под вопросом. Показано, что преимущество GPU по сравнению с CPU возрастает с увеличением числа уравнений. Например, GPU Tesla C2075 для 1920 уравнений в системе работает в 4 раза быстрее CPU AMD, а для 11008 уравнений в 7 раз.

Ключевые слова: GPU Tesla C2075, MAGMA, CUBLAS, CUDA, графический процессор,

д.т.н., проф. Мясіщев О.А., д.т.н., проф. Лєнков С.В.
к.т.н., доц. Джулій В.М., к.т.н., доц. Муляр І.В.

ВИКОРИСТАННЯ GPU NVIDIA ПРИ ВИРІШЕННІ ЗАВДАНЬ ЛІНІЙНОЇ АЛГЕБРИ

В роботі досліджується доцільність застосування графічних процесорів при вирішенні систем лінійних рівнянь і розрахунку матричного множення у порівнянні зі звичайними багатоядерними процесорами. Розглядаються особливості використання і проблеми установки бібліотеки MAGMA. Для проведення обчислювальних експериментів розглянуті дві системи. У кожній з них встановлено шестиядерний процесор (CPU) AMD. У першій системі використаний графічний процесор (GPU) Tesla C2075, в другій GeForce GTX 480 фірми NVIDIA. GPU виконують роль обчислювальних прискорювачів для вирішення систем лінійних рівнянь і матричного множення. Причому в першому випадку розрахунок виконується з урахуванням розпаралелювання по 6-ядер процесора з використанням бібліотек MPI, ScaLAPACK і ATLAS. У другому і третьому випадках - розпаралелюванням по ядрах GPU Tesla C2075 і GeForce GTX 480 з використанням технології CUDA. Обчислювальні системи працюють під управлінням операційної системи Linux Ubuntu. На них встановлені компілятори фортран і C ++ з перерахованими вище бібліотеками для 6-и ядер-

ного процесора. Для програмування на GPU Tesla C2075 і GeForce GTX 480 інстальовані відеодрайвер nvidia і програмне забезпечення CUDA Toolkit. Встановлено, що продуктивність GPU GeForce GTX 480 і GPU Tesla C2075 вище продуктивності CPU AMD приблизно в 3.5 і 6.3 разів відповідно для чисел з подвійною точністю. А продуктивність GPU GeForce GTX 480 в 1.3 рази вище продуктивності GPU Tesla C2075 для чисел з одинарної точністю. Показано, що для досягнення максимальної продуктивності GPU NVIDIA CUDA необхідно використання бібліотек MAGMA або CUBLAS, які дають прискорення розрахунків приблизно в 6.4 рази в порівнянні з традиційним способом програмування. Встановлено, що при вирішенні систем рівнянь на 6-й ядерному CPU вдається досягти максимального прискорення в 3.24 рази в порівнянні з розрахунками на 1-му ядрі при використанні бібліотек ScaLAPACK і ATLAS. Шестиразового прискорення отримати не вдається. Тому ефективне використання процесорів з великою кількістю ядер з розглянутими бібліотеками стоїть під питанням. Показано, що перевага GPU у порівнянні з CPU зростає зі збільшенням числа рівнянь. Наприклад, GPU Tesla C2075 для 1920 рівнянь в системі працює в 4 рази швидше CPU AMD, а для 11008 рівнянь в 7 разів.

Ключові слова: GPU Tesla C2075, MAGMA, CUBLAS, CUDA, графічний процесор, NVIDIA Tesla V100, SMP система, MPI, ScaLAPACK, ATLAS.