DOI 10.31891/2307-5732-2019-271-2-102-110
УДК 512.683

А. А. МЯСИЩЕВ, В. М. ПОЛОЗОВА
Хмельницкий национальный университет

ПРОИЗВОДИТЕЛЬНОСТЬ GPU И CPU ДЛЯ МАТРИЧНОГО УМНОЖЕНИЯ

В работе исследуется целесообразность применения графических процессоров при решении задач матричного умножения по сравнению с обычными многоядерными процессорами. Указываются 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 задач 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 Linux Ubuntu. На них установлены компиляторы фортран и C++ с перечисленными выше библиотеками для 6-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 CPU AMD примерно в 3.5 и 6.3 раз соответственно для чисел с двойной точностью. А производительность GPU GeForce GTX 480 в 1.3 раза выше производительности GPU Tesla C2075 для чисел с одинарной точностью. Показано, что для достижения максимальной производительности GPU NVIDIA CUDA необходимо использование библиотек MAGMA или CUBLAS, которые дают ускорение расчетов примерно в 6.4 раза по сравнению с традиционным способом программирования с использованием глобальной памяти. Рассмотрено решение задачи матричного умножения с использованием разделяемой памяти. Показано, что в этом случае производительность GPU незначительно ниже по сравнению с использованием библиотек MAGMA. Показано, что производительность GPU при матричном умножении с использованием CUBLAS для GeForce GTX 480 практически равна пиковой производительности при двойной точности вычислений (пиковая - 168.1, полученная 165.1 Гигафлопс). Для GPU Tesla C2075 пиковая производительность почти в два раза выше полученной (пиковая – 515.2, полученная 300.6 Гигафлопс). Это указывает на недостаточную эффективность библиотеки CUBLAS и MAGMA для GPU Tesla.

Ключевые слова: GPU Tesla C2075, MAGMA, CUBLAS, CUDA, графический процессор, NVIDIA Tesla V100, SMP система, MPI, ScaLAPACK, ATLAS

A. A. MYASISCHEV, V. M. POLOZOVA
Khmelnytskyi National University

GPU AND CPU PERFORMANCE FOR MATRIX MULTIPLICATION

The paper investigates the expediency of using graphics processors when solving problems of matrix multiplication compared to conventional multi-core processors. Indicates the features of the use and installation problems of the library MAGMA. To carry out computational experiments, two systems were considered. Each of them has a six-core AMD processor. The first system uses a graphics processor (GPU) Tesla C2075, the second GeForce GTX 480 from NVIDIA. GPUs play the role of computational accelerators for solving problems of matrix multiplication. Moreover, in the first case, the calculation is performed taking into account parallelization across the 6 cores of the processor using the MPI, ScaLAPACK and ATLAS libraries. In the second and third cases - parallelization across the cores of the Tesla C2075 GPU and GeForce GTX 480 using the CUDA technology. Computing systems run under the Linux Ubuntu operating system. The compilers Fortran and C ++ are installed on them with the libraries listed above for a 6-core processor. For programming on the Tesla C2075 GPU and GeForce GTX 480, the nvidia video driver and CUDA Toolkit software are installed. It has been established that the performance of the GPU GeForce GTX 480 and the GPU Tesla C2075 is higher than the performance of the AMD CPU by approximately 3.5 and 6.3 times, respectively, for double-precision numbers. And the performance of the GPU GeForce GTX 480 is 1.3 times higher than the performance of the GPU Tesla C2075 for numbers with single precision. It is shown that in order to achieve maximum performance of the NVIDIA CUDA GPU, it is necessary to use the MAGMA or CUBLAS libraries, which accelerate the calculations by about 6.4 times compared to the traditional programming method using global memory. The solution of the problem of matrix multiplication using shared memory is considered. It is shown that in this case the performance of the GPU is not significantly lower compared with the use of the MAGMA libraries. It is shown that GPU performance with matrix multiplication using CUBLAS for GeForce GTX 480 is almost equal to peak performance with double computational accuracy (peak - 168.1, obtained 165.1 Gigaflops). For the Tesla C2075 GPU, the peak performance is almost twice as high as obtained (peak - 515.2, obtained 300.6 Gigaflops). This indicates a lack of effectiveness of the CUBLAS and MAGMA libraries for the Tesla GPU.

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

Постановка задачи

Задачи теории упругости, пластичности сводятся к системе дифференциальных уравнений в напряжениях и скоростях. Обычно они решаются численно с использованием метода конечных элементов. В свою очередь, этот метод сводится к построению так называемых матриц жесткости и решению систем линейных уравнений, содержащей тысячи, десятки тысяч и даже сотни тысяч неизвестных. Для упругопластических и пластических задач решение систем уравнений приходится выполнять сотни и тысячи раз для выявления конечной конфигурации деформируемого материала. Для этих целей требуются вычислительные системы высокой производительности. В настоящее время широкое распространение получили компьютеры с многоядерными процессорами, представляющие собой параллельную

вычислительную систему с общей памятью (SMP-система). С появлением чипа NVIDIA восьмого поколения G80 (2007 г.) возникла программно-аппаратная архитектура CUDA, позволяющая производить вычисления с использованием графических процессоров NVIDIA. Эта технология представляет собой, как и в случае с процессором, параллельную вычислительную систему, в составе которой работает сотни процессорных ядер. Проблема заключается в том, чтобы распараллелить перемножение матриц так, чтобы получить максимальную загрузку многоядерного процессора CPU и процессорных ядер GPU и сопоставить, насколько отличается производительность GPU от CPU для решения подобных задач при использовании чисел двойной точности.

Изложение основного материала работы

Известны многочисленные работы, посвященные исследованию производительности работы параллельных систем, как на базе кластерных систем, так и многоядерных процессоров [1, 2, 4, 5, 7–9]. Например, в работах [6, 7, 9] представлены решения ряда задач на GPU NVIDIA. Однако не был проведен подробный анализ возможностей GPU NVIDIA при использовании как простых методов программирования, так и методов программирования на базе библиотек программ cuBLAS (CUDA Basic Linear Algebra Subroutines), MAGMA (Matrix Algebra on GPU and Multicore Architectures) [10, 11] для решения задач матричного умножения с числами одинарной и двойной точности.

В настоящее время компания Nvidia представила ускоритель на базе архитектуры Volta - Tesla V100 [3]. Данное устройство располагает высоким уровнем вычислительной мощности, составляющим около 15 Тфлопс в операциях одинарной точности и 7,5 Тфлопс в двойной, что позволяет его использовать в современных суперкомпьютерных системах [14]. Также Nvidia представила новое семейство графических процессоров GeForce RTX 20 Series [15] основанных на новой архитектуре Turing, вычислительные возможности которых для операций с одинарной точностью составляют около 12 Тфлопс, двойной – 0.37 Тфлопс. Однако в настоящее время все еще актуальными являются бюджетные GPU на базе NVIDIA Tesla C2075 и видеокарте NVIDIA GeForce GTX 480, которые установлены на многих рабочих станциях, используемых для высокопроизводительных вычислений. Поэтому в работе будет уделено внимание проведению вычислительных экспериментов для этих GPU, а в качестве многоядерного процессора с SMP архитектурой будет рассмотрен компьютер с процессором AMD Phenom II X6 1090T (CPU, 3.2 ГГц).

Целью работы является исследование целесообразности применения графических процессоров при решении задачи матричного умножения по сравнению с обычными многоядерными процессорами. Рассматриваются особенности использования и проблемы установки библиотеки MAGMA.

Для достижения цели использовались две вычислительные системы:

- в первой установлен процессор (CPU) AMD Phenom II X6 1090T с частотой 3.2 ГГц и графический процессор (GPU) Tesla C2075 фирмы «NVIDIA»;
- во второй также установлен процессор (CPU) AMD Phenom II X6 1090T с частотой 3.2 ГГц, но с графическим процессором (GPU) GeForce GTX 480 фирмы «NVIDIA».

Для решения задач проводился вычислительный эксперимент в котором определялась производительность в GFlop/s и время расчета на двух указанных выше вычислительных системах задачи матричного умножения. Использовались тестовые примеры с библиотек ScaLAPACK, ATLAS, MAGMA, CUBLAS. Также использовались собственные программы на базе рассмотренных библиотек и традиционных способов программирования.

В первом случае расчет будет выполняться с учетом распараллеливания по 6-ядрам процессора с использованием библиотек MPI, ScaLAPACK и ATLAS [1, 2, 4]. Во втором и третьем случаях – распараллеливанием по ядрам GPU Tesla C2075 и GeForce GTX 480 с использованием технологии CUDA [7]. Вычислительные системы работают под управлением операционной системы Linux Ubuntu. На них установлены компиляторы фортран F77, gfortran с перечисленными выше библиотеками для 6-ядерного процессора. Для программирования на GPU Tesla C2075 и GeForce GTX 480 инсталлированы видеодрайвер nvidia и программное обеспечение CUDA Toolkit с сайта http://developer.nvidia.com/cuda-toolkit-archive. Последовательность установки программного обеспечения для GPU представлена в источнике [12]. Установка библиотек MPI, ScaLAPACK, ATLAS под ОС Linux Ubuntu представлена в работе [5] для системы с 4-ядерным процессором CORE 2 QUAD PENTIUM Q6600 2.4GHZ, поэтому здесь не рассматривается. Установка библиотеки MAGMA и решением проблем, которые возникают при ее установке рассмотрены в работе [6].

Как указано в работе [6] компиляция тестовых программ testing_sgemm.cpp и testing_dgemm.cpp перемножения матриц с числами одинарной и двойной точности соответственно выполняется командными строками:

```
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
```

При запуске этих программ выполняется сопоставление производительностей расчета для библиотек cuBLAS и MAGMA. Производительность фиксируется в GFlop/s. Перед компиляцией этих тестовых программ необходимо откорректировать параметры istart = 1024 и iend = 10240, которые задают

начальные и конечные значения размерностей матриц 1024 и 10240. Слишком большие их значения могут не поместиться в глобальную память GPU. Рассмотрим результаты работы программ testing_sgemm.cpp и testing_dgemm.cpp для 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
```

Сопоставляя результаты, видим, что чем больше размер матрицы, тем быстрее работает библиотека CUBLAS. Однако разница в производительности двух библиотек несущественная. Резко падает производительность при переходе от чисел с одинарной точности к двойной точности (примерно в 5.1 раза). Расчеты для GPU GeForce GTX 480 были выполнены режима Fermi family.

Рассмотрим аналогичные результаты расчета для GPU Tesla C2075 для режима 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

Видно, что библиотека MAGMA имеет производительность немного выше CUBLAS для матриц большого размера. Производительность при переходе от чисел с одинарной точности к двойной точности для этого GPU падает примерно в 2 раза как в случае с многоядерными 64-разрядными процессорами. В работе выполнены сравнительные расчеты и для режима компиляции Tesla family.

Проанализируем, насколько выше эффективность работы GPU по технологии NVIDIA CUDA по сравнению с традиционным способом написания параллельных программ. Для этого сопоставим эффективность работы программ умножения матриц с использованием глобальной памяти GPU, разделяемой памяти, которые составлены по методике, изложенной в работах [7, 9] с программой, использующей библиотеки MAGMA и CUBLAS.

Для составления программы с использованием глобальной памяти GPU рассмотрим программную модель GPU для компилятора Си [9]. Верхний уровень ядра, называемый сеткой (grid) состоит из блоков. В свою очередь блоки представляют собой либо одноразмерную, либо двухразмерную сеть нитей. Вышесказанное может быть проиллюстрировано рис. 1. Номер нити в блоке или номер блока в grid может быть переменной типа int или dim3.

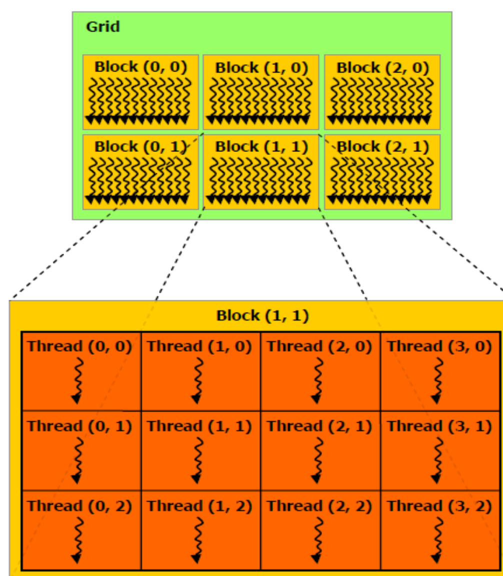

Рис. 1. Программная модель GPU

Число нитей, входящих в блок определяется встроенной переменной blockDim. Индекс нити внутри блока определяется переменной threadIdx, а индекс блока внутри grid – переменной blockIdx. Нити в блоке являются непосредственными исполнителями вычислений. Нить является 3-компонентным вектором, т.е. может идентифицироваться, используя одно размерный, двух размерный и трех размерный индекс нити, образуя в свою очередь одноразмерный, двух размерный и трех размерный блок нитей. Существует ограничение количества нитей на один блок, которое не может превышать 1024 нити. Расширение Си CUDA позволяет вызвать функцию, называемую ядром так, что она будет параллельно выполнять N разных нитей CUDA. Такие функция декларируется спецификатором __global__.

Ниже рассматриваются программы произведения квадратных заполненных матриц, которые использованы для сопоставления производительности CPU и GPU. Программы разбиты на три класса в соответствии с типом используемой памяти на GPU и библиотеки расчета произведения [7, 9]:

1. При расчете произведения матрицы размещаются в глобальной памяти GPU.
2. Матрицы размещаются в разделяемой памяти GPU.
3. Расчет выполняется программой, использующей библиотеки MAGMA и CUBLAS.

<u>Первая программа</u>. Предполагается, что матрицы a, b и c размещаются в одномерных массивах. Ниже представлен текст программы с пояснениями:

```
#include <stdio.h>
#define BLOCK 16 //Устанавливаем размер блока
__global__ void mulMatr(float* a, float* b, float* c, int n) // Функция умножения двух матриц
{ //Получаем id текущей нити.
  int i = threadIdx.y+blockIdx.y*blockDim.y; int j = threadIdx.x+blockIdx.x*blockDim.x;
     float sum=0.0f;  //Расчитываем результат.
     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;
 float* a = new float[M]; float* b = new float[M]; float* c = new float[M]; //Выделяем память под вектора
  for (int i = 0; i < N; i++){//Инициализируем значения векторов
   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];}}
  float* deva;float* devb;float* devc; //Указатели на память в видеокарте
//Выделяем память для векторов на видеокарте
  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 );
//Копируем данные в память видеокарты
  cudaMemcpy(deva, a, sizeof(float) * M, cudaMemcpyHostToDevice);
  cudaMemcpy(devb, b, sizeof(float) * M, cudaMemcpyHostToDevice);
//Выполняем вызов функции ядра
dim3 threads = dim3(BLOCK,BLOCK); // Количество нитей в блоке
dim3 blocks  = dim3(N/BLOCK,N/BLOCK); // Количество блоков в grid-е
cudaEventRecord(start, 0); // привязываем событие к началу выполнения ядра
mulMatr<<<blocks, threads>>>(deva, devb, devc,N);
cudaEventRecord(stop, 0); // привязываем событие к концу выполнения ядра. Получаем результат расчета
 cudaMemcpy(c, devc, sizeof(float) * M, cudaMemcpyDeviceToHost);
cudaEventSynchronize(stop); //Дожидаемся выполнение ядра, синхронизируя по событию stop
cudaEventElapsedTime (&gpuTime,start,stop);//Запрашиваем время между start, stop
  //Результаты расчета
 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)]);
  // Высвобождаем ресурсы
  cudaEventDestroy(start); cudaEventDestroy(stop); cuda-Free(deva);cudaFree(devb);cudaFree(devc);
  delete[] a; a = 0;delete[] b; b = 0;delete[] c; c = 0; }
```

Вторая программа. В функции ядра используется разделяемая память. Здесь принимается, что из-за малого размера разделяемой памяти, исходные квадратные матрицы разбиваются на блочные матрицы размером 16х16 элементов (BLOCK_SIZE 16), как в работе [5]. Текст представленной программы, которая выполняется на GPU (__global__ void matrixMul), соответствует тексту программы-примера, которая поставляется с сайта http://developer.nvidia.com/cuda-toolkit-32-downloads под именем matrixMul_kernel.cu.

```
#include <stdio.h>
#define BLOCK_SIZE 16
#define AS(i, j) As[i][j]
#define BS(i, j) Bs[i][j]
__global__ void
matrixMul( double* C, double* A, double* B, int wA, int wB)
{     int bx = blockIdx.x;  int by = blockIdx.y; // Block index
    int tx = threadIdx.x; int ty = threadIdx.y; // Thread index
    int aBegin = wA * BLOCK_SIZE * by;     // Index of the first sub-matrix of A processed by the block
    int aEnd   = aBegin + wA - 1; // Index of the last sub-matrix of A processed by the block
    int aStep  = BLOCK_SIZE;    // Step size used to iterate through the sub-matrices of A
    int bBegin = BLOCK_SIZE * bx;    // Index of the first sub-matrix of B processed by the block
    int bStep  = BLOCK_SIZE * wB; // Step size used to iterate through the sub-matrices of B
    double Csub = 0; // Csub is used to store the element of the block sub-matrix  that is computed by the thread
    // Loop over all the sub-matrices of A and B required to compute the block sub-matrix
    for (int a = aBegin, b = bBegin; a <= aEnd; a += aStep, b += bStep) {
        // Declaration of the shared memory array As used to store the sub-matrix of A
        __shared__ double As[BLOCK_SIZE][BLOCK_SIZE];
        // Declaration of the shared memory array Bs used to store the sub-matrix of B
        __shared__ double Bs[BLOCK_SIZE][BLOCK_SIZE];
        // Load the matrices from device memory to shared memory; each thread loads one element of each matrix
        AS(ty, tx) = A[a + wA * ty + tx]; BS(ty, tx) = B[b + wB * ty + tx];
        __syncthreads();  // Synchronize to make sure the matrices are loaded
        // Multiply the two matrices together; each thread computes one element of the block sub-matrix
        for (int k = 0; k < BLOCK_SIZE; ++k) Csub += AS(ty, k) * BS(k, tx);
// Synchronize to make sure that the preceding computation is done before loading two new sub-matrices of A and B in // the next iteration
 __syncthreads(); }
    // Write the block sub-matrix to device memory; each thread writes one element
    int c = wB * BLOCK_SIZE * by + BLOCK_SIZE * bx; C[c + wB * ty + tx] = Csub;  }
__host__ int main()
{ int N; int M; double mf=0.0f;
 printf ( "Input N->");  scanf ( "%d",&N); printf ( "Matrix = %dx%d elements\n", N,N );  M=N*N;
//Выделяем память под вектора
  double* a = new double[M]; double* b = new double[M]; double* c = new double[M];
//Инициализируем значения векторов
```

```
  for (int i = 0; i < N; i++){ for (int j = 0; j < N; j++) {  a[i*N+j] = 1.0*((i+1)+2*(j+1));
b[i*N+j] = 1.0/a[i*N+j]; } }
  double* deva; double* devb; double* devc; //Указатели на память в видеокарте
//Выделяем память для векторов на видеокарте
  cudaMalloc((void**)&deva, sizeof(double) * M); cudaMalloc((void**)&devb, sizeof(double) * M);
  cudaMalloc((void**)&devc, sizeof(double) * M);
// create cuda event handle
cudaEvent_t start, stop; float gpuTime=0.0f; cudaEventCreate ( &start ); cudaEventCreate ( &stop
);
//Копируем данные в память видеокарты
cudaMemcpy(deva, a, sizeof(double) * M, cudaMemcpyHostToDevice);
cudaMemcpy(devb, b, sizeof(double) * M, cudaMemcpyHostToDevice);
//Выполняем вызов функции ядра
dim3 threads = dim3(BLOCK_SIZE,BLOCK_SIZE); dim3 blocks  = dim3(N/BLOCK_SIZE,N/BLOCK_SIZE);
cudaEventRecord(start, 0); // привязываем событие к началу выполнения ядра
matrixMul<<<blocks, threads>>>(devc, deva, devb,N,N); cudaEventRecord(stop, 0); // привязываем
событие
//к концу выполнения ядра
cudaMemcpy(c, devc, sizeof(double) * M, cudaMemcpyDeviceToHost); //Получаем результат расчета
cudaEventSynchronize(stop); //Дожидаемся выполнение ядра, синхронизируя по событию stop
cudaEventElapsedTime (&gpuTime,start,stop);//Запрашиваем время  между start, stop
//Результаты расчета
 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)]);
 // Высвобождаем ресурсы
 cudaEventDestroy(start);        cudaEventDestroy(stop);     cudaFree(deva);     cudaFree(devb);
cudaFree(devc);
  delete[] a; a = 0; delete[] b; b = 0; delete[] c; c = 0;}
```

Третья программа. Использует библиотечные функции.

```
#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 elements\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]); }
```

Последняя программа написаны для чисел с одинарной точностью. Для чисел с двойной точностью необходимо поменять тип данных float на double, а функцию magmablas_sgemm необходимо заменить на magmablas_dgemm. Программа использует библиотеки MAGMA. Для использования CUBLAS необходимо вместо функции magmablas_sgemm ввести в программу функцию cublasSgemm.

В таблице 1 сведены результаты расчетов по представленным выше программам для GPU при двойной точности вычислений. Также представлены результаты и для CPU с использованием библиотек ScaLAPACK и ATLAS по методу, изложенному в работе [5]. Результаты представлены в виде дроби: числитель – время счета в секундах, знаменатель – производительность в Гигафлопсах в секунду. Производительность определялась как отношение числа операций с плавающей точкой при матричном произведении к затраченному времени. Число операций в программах определяется по выражению: op=N*N(2*N-1), где N – число строк или столбцов квадратной матрицы.

Сопоставление данных в табл. 1, 2 с результатами тестовых программ показало их близкое сходство. Производительность процессора соизмерима с производительностью программы для GPU Tesla C2075, написанной с использованием глобальной памяти [7] и примерно в 6,3 раза ниже для GPU Tesla

C2075 для программ с использованием библиотек CUBLAS и MAGMA. В случае компиляции в режиме Tesla family библиотека MAGMA существенно проигрывает по производительности библиотеке CUBLAS. Так для чисел с двойной точностью примерно в 1.8 раза.

Таблица 1

Результаты расчетов по представленным выше программам для GPU при двойной точности вычислений

Мат. NxN	CPU 6 ядер	GPU GeForce GTX 480			GPU Tesla C2075		
		Глоб.	Разд.	cublas	Глоб.	Разд.	cublas
1024	0.062/34.6	0.034/61.9	0.018/122.1	0.014/155.7	0.045/47.3	0.022/96.0	0.008/284.4
2048	0.407/42.2	0.282/60.8	0.140/123.0	0.105/163.0	0.375/45.9	0.178/96.4	0.058/295.8
3072	1.234/47.0	0.919/63.1	0.471/123.1	0.352/164.6	1.233/47.0	0.600/96.7	0.194/299.2
4096	2.870/47.9	2.213/62.1	1.141/120.5	0.835/164.5	2.963/46.4	1.455/94.5	0.458/300.1
5120	5.687/47.2	4.253/63.1	2.181/123.0	1.626/165.1	5.715/47.0	2.781/96.5	0.893/300.5
6144	- / -	7.482/62.0	3.771/123.0	2.810/165.1	10.025/46.3	4.805/96.5	1.543/300.6

Таблица 2

Результаты расчетов для режимов компиляции Fermi family и Tesla family

Матрица NxN	Процессор AMD Phenom II X6 1090T(6 ядер)	GPU Tesla C2075				
		Глобальная память	Компиляция Tesla		Компиляция Fermi	
			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

Выводы

1. Процесс установки библиотек MAGMA не является самонастраивающимся. Часто приходится вручную делать корректировки используемых библиотек при компиляции и изменять тексты программ. Поэтому установка библиотек требует достаточной квалификации системного программиста.

2. Библиотеки MAGMA и CUBLAS показали примерно одинаковую производительность для матричного произведения как для чисел с одинарной, так и с двойной точностью. Однако для GPU GeForce GTX 480 резко падает производительность при переходе от чисел с одинарной точности к двойной точности (примерно в 5.1 раза). Для GPU Tesla C2075 падение производительности не превышает 2,1 раза. Таким образом, GPU Tesla C2075 более подходит для решения сложных задач, требующих расчетов с двойной точностью, который также имеет объем глобальной памяти 6 Гбайт (GPU GeForce GTX 480 – 1,5 ГБайт).

3. Производительность GPU GeForce GTX 480 и GPU Tesla C2075 выше производительности CPU AMD Phenom II X6 1090T примерно в 3.5 и 6.3 раз соответственно для чисел с двойной точностью. А производительность GPU GeForce GTX 480 в 1.3 раза выше производительности GPU Tesla C2075 для чисел с одинарной точностью. Поэтому для небольших задач, требующих памяти не более 1.5 Гбайт и расчетов с одинарной точностью целесообразнее использовать GPU GeForce GTX 480 как недорогое и очень эффективное решение.

4. Для достижения максимальной производительности GPU NVIDIA CUDA необходимо обязательно использовать библиотеки MAGMA или CUBLAS, которые дают ускорение рассмотренных расчетов примерно в 6.4 раза по сравнению с использованием глобальной памяти (традиционный способ программирования). Программирование с использованием разделяемой памяти требует высокой квалификации программиста.

5. Преимущество GPU по сравнению с CPU возрастает с размерности матрицы.

6. Производительность GPU при матричном умножении с использованием CUBLAS для GeForce GTX 480 практически равна пиковой производительности при двойной точности вычислений (пиковая – 168.1, полученная 165.1 Гигафлопс). Для GPU Tesla C2075 пиковая производительность почти в два раза выше полученной (пиковая – 515.2, полученная 300.6 Гигафлопс). Это указывает на недостаточную эффективность библиотеки CUBLAS и MAGMA для GPU Tesla

7. Стоимость GPU GeForce GTX 480 примерно в 8 раз меньше стоимости GPU Tesla. Поэтому, исходя из соотношения стоимость – производительность выгоднее использование для небольших расчетов, т.е. требующих памяти до 1.5 Гбайт, GPU GeForce GTX 480. Особенно выгодно его использование для расчетов с одинарной точностью, т.к. GPU GeForce GTX 480 работает в 8.4 раза быстрее CPU AMD Phenom II X6 1090T, а GPU Tesla C2075 лишь в 6.5 раза (это показали расчеты по этим же программам, которые были переписаны для чисел с одинарной точностью).

8. Рассмотренные способы решения задач матричного умножения справедливы для использования в вычислительной системе только одного GPU. Однако неизвестно, насколько эффективно распараллеливание

этой задачи на нескольких установленных GPU в многоядерной системе.

Литература

1. The ScaLAPACK Project. 2017 [Электронный ресурс]. – Режим доступа : http://www.netlib.org
2. Automatically Tuned Linear Algebra Software (ATLAS). 2016 [Электронный ресурс]. – Режим доступа : http://math-atlas.sourceforge.net/
3. NVIDIA TESLA V100 TENSOR CORE GPU. 2018 [Электронный ресурс]. – Режим доступа : https://www.nvidia.com/en-us/data-center/tesla-v100/
4. Антонов А.С. Параллельное программирование с использованием технологии MPI : учебное пособие / Антонов А.С. – М. : Изд-во МГУ, 2004. – 71 с.
5. Мясищев А.А. Достижение наибольшей производительности перемножения матриц на системах с многоядерными процессорами. Т. 3: Теорія та методика навчання інформатики / Мясищев А.А. – Кривий Ріг : Видавничий відділ НметАУ, 2010. – 303 с.
6. Мясіщев О.А. Оцінка продуктивності GPU NVIDIA CUDA при вирішенні задач матричного множення / О.А. Мясіщев // Вимірювальна та обчислювальна техніка в технологічних процесах. – Хмельницький, 2012. – № 1. – С. 73–79.
7. Параллельные вычисления на GPU. Архитектура и программная модель GPU : учеб. пособие / А.В. Боресков и др. ; предисл.: В.А. Садовничий. – М. : Издательство Московского университета, 2012. – 336 с., илл. – (Серия "Суперкомпьютерное образование").
8. Гергель В.П. Теория и практика параллельных вычислений. 2016 [Электронный ресурс]. – Режим доступа : https://www.twirpx.com/file/1978282/
9. Параллельные вычисления на GPU. Архитектура и программная модель CUDA : учеб. пособие / Боресков А.В., Харламов А.А., Марковский Н.Д. и др. – М. : Издательство Московского университета, 2012. – 336 с.
10. Matrix Algebra on GPU and Multicore Architectures. 2018 [Электронный ресурс]. – Режим доступа : http://icl.cs.utk.edu/magma/index.html
11. CUBLAS (NVIDIA CUDA Basic Linear Algebra Subroutines). 2018 [Электронный ресурс]. – Режим доступа : http://developer.nvidia.com/cublas
12. Установка CUDA Toolkit и драйвера NVIDIA для разработчиков. 2012 [Электронный ресурс]. – Режим доступа : http://forum.ubuntu.ru/index.php?topic=114802.0
13. Мясищев А.А. Вычислительные возможности микроконтроллеров STM32F4/ А.А. Мясищев, С.В. Ленков // Збірник наукових праць Військового інституту Київського національного університету імены Тараса Шевченка. – Київ, 2016. – Випуск № 51. – С. 185–192.
14. Top 500 the list. 2018 [Электронный ресурс]. – Режим доступа : https://www.top500.org/lists/2018/06/
15. NVIDIA TURING. 2018 [Электронный ресурс]. – Режим доступа : https://www.nvidia.com/ru-ru/geforce/turing/
16. Myasischev A.A. Computing capabilities stm32f429i-disco for matrix multiplication / A.A. Myasischev // Materials of the XI International scientific and practical conference. – 2015. – Vol 17. Sheffild. – С. 51–57.
17. Мясіщев О.А. Ефективність використання GPU NVIDIA при вирішенні систем лінійних рівнянь / О.А. Мясіщев // Зб. наук. праць Військового інституту Київського НУ ім. Тараса Шевченко. – К. : ВІКНУ, 2012. – Вип. 38. – С. 76–80.
18. Мясищев А.А. Сопоставление производительностей GPU и CPU для матричного умножения с двойной точностью / А.А. Мясищев // Теорія та методика навчання математики, фізики, інформатики : збірник наукових праць : в 3-х т. Т. 3: Теорія та методика навчання інформатики. – Кривий Ріг : Видавничий відділ НМетАУ, 2012. – Випуск Х. – С. 99–108.
19. Мясищев А.А. Сопоставление производительностей GPU TESLA C2075, GEFORCE 480 GTX с 6-ядерным CPU AMD при решении систем линейных уравнений / А.А. Мясищев // Материалы междунар. НПК «Ключевые проблемы современной науки – 2012». – София : Бял ГРАД-БГ, 2012. – Том 30. – С. 28–40.

References

1. The ScaLAPACK Project. 2017 [Jelektronnyj resurs]. – Rezhim dostupa : http://www.netlib.org
2. Automatically Tuned Linear Algebra Software (ATLAS). 2016 [Jelektronnyj resurs]. – Rezhim dostupa : http://math-atlas.sourceforge.net/
3. NVIDIA TESLA V100 TENSOR CORE GPU. 2018 [Jelektronnyj resurs]. – Rezhim dostupa : https://www.nvidia.com/en-us/data-center/tesla-v100/
4. Antonov A.S. Parallel'noe programmirovanie s ispol'zovaniem tehnologii MPI : uchebnoe posobie / Antonov A.S. – M. : Izd-vo MGU, 2004. – 71 s.
5. Mjasishhev A.A. Dostizhenie naibol'shej proizvoditel'nosti peremnozhenija matric na sistemah s mnogojadernymi processorami. T. 3: Teorija ta metodika navchannja informatiki / Mjasishhev A.A. – Krivij Rig : Vidavnichij viddil NmetAU, 2010. – 303 s.
6. Mjasishhev O.A. Ocinka produktivnosti GPU NVIDIA CUDA pri virishenni zadach matrichnogo mnozhennja / O.A. Mjasishhev // Vimirjuval'na ta obchisljuval'na tehnika v tehnologichnih procesah. – Hmel'nic'kij, 2012. – № 1. – S. 73–79.

7. Parallel'nye vychislenija na GPU. Arhitektura i programmnaja model' GPU : ucheb. posobie / A.V. Boreskov i dr. ; predisl.: V.A. Sadovnichij. – M. : Izdatel'stvo Moskovskogo universiteta, 2012. – 336 s., ill. – (Serija "Superkomp'juternoe obrazovanie").

8. Gergel' V.P. Teorija i praktika parallel'nyh vychislenij. 2016 [Jelektronnyj resurs]. – Rezhim dostupa : https://www.twirpx.com/file/1978282/

9. Parallel'nye vychislenija na GPU. Arhitektura i programmnaja model' CUDA : ucheb. posobie / Boreskov A.V., Harlamov A.A., Markovskij N.D. i dr. – M. : Izdatel'stvo Moskovskogo universiteta, 2012. – 336 s.

10. Matrix Algebra on GPU and Multicore Architectures. 2018 [Jelektronnyj resurs]. – Rezhim dostupa : http://icl.cs.utk.edu/magma/index.html

11. CUBLAS (NVIDIA CUDA Basic Linear Algebra Subroutines). 2018 [Jelektronnyj resurs]. – Rezhim dostupa : http://developer.nvidia.com/cublas

12. Ustanovka CUDA Toolkit i drajvera NVIDIA dlja razrabotchikov. 2012 [Jelektronnyj resurs]. – Rezhim dostupa : http://forum.ubuntu.ru/index.php?topic=114802.0

13. Mjasishhev A.A. Vychislitel'nye vozmozhnosti mikrokontrollerov STM32F4/ A.A. Mjasishhev, S.V. Lenkov // Zbirnik naukovih prac' Vijs'kovogo institutu Kiïvs'kogo nacional'nogo universitetu imeny Tarasa Shevchenka. – Kiïv, 2016. – Vipusk № 51. – S. 185–192.

14. Top 500 the list. 2018 [Jelektronnyj resurs]. – Rezhim dostupa : https://www.top500.org/lists/2018/06/

15. NVIDIA TURING. 2018 [Jelektronnyj resurs]. – Rezhim dostupa : https://www.nvidia.com/ru-ru/geforce/turing/

16. Myasischev A.A. Computing capabilities stm32f429i-disco for matrix multiplication / A.A. Myasischev // Materials of the XI International scientific and practical conference. – 2015. – Vol 17. Sheffild. – S. 51–57.

17. Mjasishhev O.A. Efektivnist' vikoristannja GPU NVIDIA pri virishenni sistem linijnih rivnjan' / O.A. Mjasishhev // Zb. nauk. prac' Vijs'kovogo institutu Kiïvs'kogo NU im. Taras Shevchenko. – K. : VIKNU, 2012. – Vip. 38. – S. 76–80.

18. Mjasishhev A.A. Sopostavlenie proizvoditel'nostej GPU i CPU dlja matrichnogo umnozhenija s dvojnoj tochnost'ju / A.A. Mjasishhev // Teorija ta metodika navchannja matematiki, fiziki, informatiki : zbirnik naukovih prac' : v 3-h t. T. 3: Teorija ta metodika navchannja informatiki. – Krivij Rig : Vidavnichij viddil NMetAU, 2012. – Vipusk X. – S. 99–108.

19. Mjasishhev A.A. Sopostavlenie proizvoditel'nostej GPU TESLA C2075, GEFORCE 480 GTX s 6-jadernym CPU AMD pri reshenii sistem linejnyh uravnenij / A.A. Mjasishhev // Materialy mezhdunar. NPK «Kljuchevye problemy sovremennoj nauki – 2012». – Sofija : Bjal GRAD-BG, 2012. – Tom 30. – S. 28–40.

Рецензія/Peer review : 2.3.2019 р. Надрукована/Printed :10.4.2019 р.
Стаття рецензована редакційною колегією