$$y_{a,b} = |a|^{-1/2} \cdot y\left(\frac{a-b}{a}\right),$$

де $a, b \in R$, $y \in L^2(R)$.

Безперервне вейвлет– перетворення записується наступним чином

$$[W_y f](a,b) = |a|^{-1/2} \int_{-\infty}^{+\infty} f(t) y\left(\frac{a-b}{a}\right) dt = \int_{-\infty}^{+\infty} f(t) y_{ab}(t) dt \cdot$$

Для позначення коефіцієнтів вейвлет– перетворення використаємо компактні позначення $W(a, b)$, $W_y f$ або $W[f]$.

Отже будь– яку функцію $f(t) \in L^2(R)$ можна одержати суперпозицією масштабних перетворень і зсувів обраного базисного вейвлету $y(t)$, тобто поданою набором вейвлетних хвиль, амплітуди яких залежать від номеру хвилі (частоти або масштабу) і параметру зсуву (часу).

Відомий ортогональний вейвлет Хаара (Haar– wavelet)

$$y^H(t) = \begin{cases} 1, & 0 \le t \le 1/2 \\ -1, & 1/2 \le t \le 1 \\ 0, & \mathbf{p}\ 0, \quad t \ge 1. \end{cases}$$

Відомі й інші види ортогональних вейвлетів, оцінка ефективності яких виконується за степенем їх локалізації в часовій і частотній областях.

Механізми частотно– часової локалізації вейвлет– перетворення розглянуті в [2].

Висновок. Перетворення нечітких значень членів множинного ряду у лінгвістичні величини істинності (фазифікація), а також одержання чіткого результату (дефазифікація) ефективно виконується з допомогою вейвлет– перетворення.

Подальші дослідження повинні бути спрямовані на створення спеціалізованих вейвлет– процесорів та їх програмне забезпечення.

Література

1. Гостев В. И. Синтез нечетких регуляторов систем автоматического управления. – Киев: из– во «Радиоаматор», 2003. – 472 с.

2. Корченко А. Г. Построение систем защиты информации на нечетких множествах. Теория и практические решения. – К.: «МК– Пресс». – 2006. – 320 с.

Надійшла до редакції
15.2.2012 р.

УДК 519.683

О.А. МЯСІЩЕВ
Хмельницький національний університет

ОЦІНКА ПРОДУКТИВНОСТІ GPU NVIDIA CUDA ПРИ ВИРІШЕННІ ЗАДАЧ МАТРИЧНОГО МНОЖЕННЯ

В роботі проведені дослідження використання графічних процесорів фірми NVIDIA, що працюють за технологією CUDA для вирішення завдань матричного множення. Розглянуто спеціальні прискорювальні бібліотеки MAGMA і CUBLAS, що дозволяють різко підвищити продуктивність розрахунків. Виконано порівняння розрахунків для CPU AMD Phenom II X6 1090T, GPU GeForce GTX 480 і Tesla C2075 для чисел одинарної і подвійної точності.

The paper investigated the use of graphics processing company NVIDIA, working on CUDA technology for solving problems of matrix multiplication. We consider a special accelerator MAGMA libraries and CUBLAS, to sharply increase the productivity calculations. The comparisons of calculations for CPU AMD Phenom II X6 1090T, GPU GeForce GTX 480 and Tesla C2075 for the number of single and double precision.

Ключові слова: GPU, NVIDIA, CUDA.

Постановка проблеми. При вирішенні задач теорії пружності, пластичності, які зводяться до вирішення систем диференціальних рівнянь в напругах і швидкостях, використовується метод кінцевих елементів. У свою чергу даний метод зводиться до побудови матриць жорсткості, перемножування матриць і вирішення систем лінійних рівнянь, що містить тисячі і навіть десятки тисяч невідомих. При вирішенні нелінійних задач теорії пружності, пластичних задач необхідно виконувати сотні і тисячі ітерацій, кожна з яких вимагає побудови своєї сітки кінцевих елементів. Все це пов'язано з необхідністю використання обчислювальних систем високої продуктивності. В даний час широке поширення одержали комп'ютери з багатоядерними процесорами. Наприклад, 6-и ядерний процесор AMD Phenom II X6 1090T (CPU), що

представляє собою паралельну обчислювальну систему із загальною пам'яттю (SMP-система). З появою чіпа NVIDIA восьмого покоління G80 (2007р.) виникла програмно-апаратна архітектура CUDA, що дозволяє робити обчислення з використанням графічних процесорів NVIDIA. Ця технологія являє собою, як і у випадку з процесором, паралельну обчислювальну систему, у складі якої працює 480 процесорних ядер, наприклад для GeForce GTX 480. Проблема полягає в тому, щоб распараллелить перемножування матриць так, щоб отримати максимальне завантаження багатоядерного процесора CPU і процесорних ядер GPU і зіставити, наскільки відрізняється продуктивність різних моделей GPU від CPU AMD Phenom II X6 1090T для завдання множення квадратних заповнених матриць при використанні чисел одинарної і подвійної точності.

Аналіз результатів попередніх досліджень. Невирішені завдання. В даний час відомі численні роботи, присвячені дослідженню продуктивності роботи паралельних систем, як на базі кластерних систем, так і багатоядерних процесорів [1,2,3,4,5]. В роботі [6] представлені вирішення низки завдань і на GPU NVIDIA. Однак не було проведено детальний аналіз можливостей сучасних GPU NVIDIA при використанні, як простих методів програмування, так і бібліотек програм cuBLAS (CUDA Basic Linear Algebra Subroutines), MAGMA [7,8] для вирішення завдань матричного перемноження з числами одинарної і подвійної точності.

Мета статті. В роботі досліджується доцільність застосування графічних процесорів при вирішенні зокрема завдань матричного множення у порівнянні зі звичайними багатоядерними процесорами. Розглядаються особливості та проблеми установки бібліотеки MAGMA (Matrix Algebra on GPU and Multicore Architectures).

Виклад основного матеріалу. Зіставимо ефективність використання для розрахунку множення квадратних матриць для одинарної і подвійної точності обчислювальну систему, в якій встановлені 6-и ядерний процесор AMD Phenom II X6 1090T (CPU, 3.2 ГГц) з відеокартою NVIDIA GeForce GTX 480 і обчислювальну систему з таким же процесором, але з графічним процесором (GPU) NVIDIA Tesla C2075. Розрахунок буде виконуватися для 3-х випадків. У першому випадку використовується тільки процесор (CPU), для якого буде виконуватися розпаралелювання по 6-ядрам з використанням бібліотек ScaLAPACK і бібліотек ATLAS (автоматично настроюється програмне забезпечення для вирішення задач лінійної алгебри). Бібліотеки ATLAS при компіляції настроюються під конкретну архітектуру процесора обчислювальної системи [4]. У другому випадку розрахунок буде виконуватися на відеокарті NVIDIA GeForce GTX 480 з використанням технології CUDA [6]. У третьому - на графічному процесорі NVIDIA Tesla C2075. Обчислювальна система налаштована на роботу з операційною системою Linux Ubuntu ver. 10.10 desktop. В системі встановлені компілятор FORTRAN - F77, бібліотеки MPI [3], ScaLAPACK [1] і ATLAS [2] для 6-и ядерного процесора. Для програмування на NVIDIA інстальовані відеодрайвер nvidia та програмне забезпечення з сайту http://developer.nvidia.com/cuda-toolkit-archive. Послідовність установки бібліотек MPI, ScaLAPACK, ATLAS під ОС Linux Ubuntu ver. 9.04 desktop докладно представлена в роботі [4] для системи з 4-х ядерним процесором CORE 2 QUAD PENTIUM Q6600 2.4GHZ, тому тут не розглядається. Установка програмного забезпечення NVIDIA CUDA з бібліотекою виконується аналогічно джерелу [9]. Тут докладніше розглянемо установку бібліотеки MAGMA 1.0 і рішенням проблем, які виникають при її установці.

Установка аналогічна для 32-і і 64-х розрядної операційної системи Linux Ubuntu 10.10 desktop. Для GPU Tesla C2075 доцільне використання 64-х розрядної системи, тому що обсяг глобальної пам'яті для нього дорівнює 6 Гбайт. Перед компіляцією MAGMA необхідно встановити бібліотеку lapack по команді:

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

Після копіювання magma_1.0.0.tar.gz з сайту http://www.cs.utk.edu/ ~ tomov/magma_1.0.0.tar.gz виконується його розархівування$gzip –d magma_1.0.0.tag.gz

```
$tar xf magma_1.0.0.tar
$cd magma_1.0.0
За допомогою редактора nano створюємо файл make.inc з наступним вмістом:
GPU_TARGET = 1
CC        = gcc
NVCC      = nvcc
FORT      = gfortran
ARCH      = ar
ARCHFLAGS = cr
RANLIB    = ranlib
OPTS      = -O3 -DADD_
FOPTS     = -O3 -DADD_ -x f95-cpp-input
NVOPTS    = --compiler-options -fno-strict-aliasing -DUNIX -O3 -DADD_
LDOPTS    = -fPIC -Xlinker -zmuldefs
LIB       = -lcblas  -llapack -lpthread -lcublas -lcudart  -lm
CUDADIR   = /usr/local/cuda
LIBDIR    =  -L/usr/local/cuda/lib -L/usr/lib
INC       = -I$(CUDADIR)/include
LIBMAGMA     = ../lib/libmagma.a
LIBMAGMABLAS = ../lib/libmagmablas.a
```

Тут необхідно звернути увагу на значення параметра GPU_TARGET. Якщо прирівняти його нулю, то бібліотека побудується для GPU Tesla family, якщо прирівняти одиниці, то це відповідає Fermi family.

GPU NVIDIA GeForce GTX 480 відповідає Fermi family. Для GPU Tesla C2075 виконаємо побудову бібліотек magma з GPU_TARGET = 1 і GPU_TARGET = 0 і порівняємо швидкодію розрахунків. Запустимо команду компіляції make all. В цьому випадку будуть створюватися бібліотеки libmagma, libmagmablas і виконається компіляція тестових прикладів. Однак тут виникають проблеми.
1. Для 32-х розрядної системи.

```
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 …
```

Тут необхідно зайти в програму control / sizeptr.c і привести її до виду

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

Тобто необхідно поміняти format% lu на format% d. Для 64-х розрядної системи компілятор gfortran помилок не видає.

2. При компіляції тестових прикладів.

Перша проблема:

```
../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'
```

Необхідно зайти в програму src / zlatrd.cpp і зробити зміни

```
    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);
```

і

```
   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);
```

Друга проблема:

```
../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'
```

Аналогічно необхідно зайти в програму src / clatrd.cpp і зробити зміни

```
//          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);
```

і

```
            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);
```

Видно, що для організації коректної зв'язку з бібліотеками fortran необхідна бібліотека libcblas замість libblasf77. Бібліотека libcblas.so встановлюється при установці liblapack і тому вона описується в файлі make.inc в параметрі LIB.

Компіляція тестових програм testing_sgemm.cpp і testing_dgemm.cpp перемножування матриць з числами одинарної і подвійної точності відповідно виконується командними рядками:

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 запуску цих програм виконується зіставлення продуктивностей розрахунку для бібліотек CUBLAS і MAGMA. Продуктивність фіксується в GFlop/s. Перед компіляцією цих тестових програм необхідно відкоригувати параметри istart = 1024 і iend = 10240, які задають початкові і кінцеві значення розмірностей матриць 1024 і 10240.Слішком великі їх значення можуть не поміститися в глобальну пам'ять GPU. Розглянемо результати роботи програм testing_sgemm.cpp і 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
```

Зіставляючи результати, бачимо, що чим більше розмір матриці, тим швидше працює бібліотека CUBLAS. Проте різниця в продуктивності несуттєва. Різко падає продуктивність при переході від чисел з одинарною точності до подвійної точності (приблизно в 5.1 рази). Розрахунки для GPU GeForce GTX 480 були виконані для параметра GPU_TARGET = 1 (Fermi family).

Розглянемо аналогічні результати розрахунку для GPU Tesla C2075 для параметра GPU_TARGET = 0 (Tesla family) і GPU_TARGET = 1 (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
 7627  7627  7627       407.07                           629.42         6.103516e-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
 7627  7627  7627       165.59                            297.30         1.136868e-13
 9533  9533  9533       166.24                            293.68         1.136868e-13
```

В цьому випадку бібліотека MAGMA істотно програє за продуктивністю бібліотеці CUBLAS як для чисел з одинарною точністю (приблизно в 1.6 рази) так і для чисел з подвійною точністю (приблизно в 1.8 рази)

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
 7627  7627  7627       638.47                           629.45         6.103516e-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
 7627  7627  7627       296.63                                 297.39         1.136868e-13
 9533  9533  9533       300.77                                 293.83         1.136868e-13
```

В цьому випадку бібліотека MAGMA має продуктивність трохи вище CUBLAS. Отже, для GPU Tesla C2075 необхідна компіляція бібліотек MAGMA з параметром GPU_TARGET = 1 (файл make.inc).

Проаналізуємо, наскільки вище ефективність роботи GPU за технологією NVIDIA CUDA в порівнянні з традиційним способом написання паралельних програм. Для цього можна порівняти ефективність роботи програми множення матриць з використанням глобальної пам'яті GPU (перша програма), яка складена за методикою, викладеною в роботі [6] з програмою, що використовує бібліотеки MAGMA і CUBLAS (друга програма). Друга програма являє спрощений варіант тестових прикладів testing_sgemm.cpp і testing_dgemm.cpp, коли розмірність матриць кратна 16.

Перша програма:

```
#include <stdio.h>
#define BLOCK 16 // Встановлюємо розмір блоку
// Функція множення двох матриць
__global__ void mulMatr(float* a, float* b, float* c, int n)
{ //Отримуємо id поточної нитки.
  int i = threadIdx.y+blockIdx.y*blockDim.y; int j = threadIdx.x+blockIdx.x*blockDim.x;
  //розраховуємо результат.
     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;
//Виділяємо пам'ять під вектора
  float* a = new float[M]; float* b = new float[M]; float* c = new float[M];
//Ініціалізуємо значення векторів
  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];}}
//Покажчики на пам'ять у відеокарті
  float* deva;float* devb;float* devc;
//Виділяємо пам'ять для векторів на відеокарті
  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 );
//Копіюємо дані в пам'ять відеокарти
  cudaMemcpy(deva, a, sizeof(float) * M, cudaMemcpyHostToDevice);
  cudaMemcpy(devb, b, sizeof(float) * M, cudaMemcpyHostToDevice);
//Виконуємо виклик функції ядра
dim3 threads = dim3(BLOCK,BLOCK); // Кількість ниток в блоці
dim3 blocks  = dim3(N/BLOCK,N/BLOCK); // Кількість блоків в grid-е
cudaEventRecord(start, 0); // Прив'язуємо подію до початку виконання ядра
mulMatr<<<blocks, threads>>>(deva, devb, devc,N);
cudaEventRecord(stop, 0); // Прив'язуємо подію до кінця виконання ядра. Отримуємо результат
розрахунку
 cudaMemcpy(c, devc, sizeof(float) * M, cudaMemcpyDeviceToHost);
cudaEventSynchronize(stop); //Чекаємо виконання ядра, синхронізуючи за подією stop
cudaEventElapsedTime (&gpuTime,start,stop);//Запитуємо час між start, stop
  //Результати розрахунку
 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)]);
  // Звільняємо ресурси
  cudaEventDestroy(start); cudaEventDestroy(stop); cudaFree(deva);cudaFree(devb);cudaFree(devc);
  delete[] a; a = 0;delete[] b; b = 0;delete[] c; c = 0; }
```

Компіляція цієї програми виконується командою:

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

Тут matrmul_g.cu - ім'я вихідного тексту програми

Друга програма:

```
#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]); }
```

Ці дві програми написані для чисел з одинарною точністю. Для чисел з подвійною точністю в них необхідно поміняти тип даних float на double. Також у другій програмі функцію magmablas_sgemm необхідно замінити на magmablas_dgemm. Друга програма використовує бібліотеки MAGMA. Для використання CUBLAS необхідно замість функції magmablas_sgemm ввести в програму функцію cublasSgemm. Компіляція програми виконується командами:

```
gcc -O3 -DADD_ -DGPUSHMEM=200 -I/usr/local/cuda/include -I../include -I../quark/include -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
```

Тут mb.cpp ім'я другої програми.

У таблиці 1 представлені результати оцінок продуктивностей за наведеними вище програмами для чисел з подвійною точністю для GPU Tesla C2075 і для CPU AMD Phenom II X6 1090T за програмами, представленим в роботі [4] з використанням бібліотек ScaLAPACK і ATLAS. У таблиці 1 чисельник - час рахунку в секундах, знаменник - продуктивність в гігафлопс в секунду. Продуктивність визначалася як відношення числа операцій з плаваючою точкою при матричному множенні до витраченому часу. Число операцій в програмах визначається за виразом: op = N * N (2 * N-1), де N-число рядків або стовпців квадратної матриці.

Таблиця 1

Матриця NxN	Процесор AMD Phenom II X6 1090T Використовується 6 ядер	GPU Tesla C2075				
		Глобальна пам'ять	Компіляція Tesla		Компіляція 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

Зіставлення даних у таблиці 1 з результатами тестових програм показало їх близьке подібність. Однак для бібліотеки CUBLAS продуктивність трохи вище, ніж для MAGMA, хоча тестові програми показують зворотну картину для компіляції Fermi family. Продуктивність процесора порівнянна з продуктивністю програми для GPU Tesla C2075, написаної з використанням глобальної пам'яті [6] і приблизно в 6,3 рази нижче для GPU Tesla C2075 для програм з використанням бібліотек CUBLAS і MAGMA.

Висновки та перспективи подальших досліджень.

1. Процес установки бібліотек MAGMA не є самоналагоджувальним. Часто доводиться вручну робити коригування використовуваних бібліотек при компіляції і змінювати тексти програм. Тому установка бібліотек вимагає достатньої кваліфікації системного програміста.

2. Велике значення на продуктивність роботи з бібліотекою MAGMA впливає значення параметра GPU_TARGET (Tesla family, Fermi family) при компіляції бібліотек для GPU Tesla C2075. Максимальна

продуктивність досягається при GPU_TARGET = 1 (Fermi Family).

3. Бібліотеки MAGMA і CUBLAS показали приблизно однакову продуктивність для матричного множення як для чисел з одинарною, так і з подвійною точністю. Однак для GPU GeForce GTX 480 різко падає продуктивність при переході від чисел з одинарною точності до подвійної точності (приблизно в 5.1 рази). Для GPU Tesla C2075 падіння продуктивності не перевищує 2,1 рази. Таким чином, GPU Tesla C2075 більше підходить для вирішення складних завдань, що вимагають розрахунків з подвійною точністю, який також має об'єм глобальної пам'яті 6 Гбайт (GPU GeForce GTX 480 - 1,5 ГБайт).

4. Продуктивність GPU GeForce GTX 480 і GPU Tesla C2075 вище продуктивності CPU AMD Phenom II X6 1090T приблизно в 3.5 та 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 одинарною точністю. Тому для невеликих завдань, що вимагають пам'яті не більше 1.5 Гбайт і розрахунків з одинарною точністю доцільніше використовувати GPU GeForce GTX 480 як недороге і дуже ефективне рішення.

5. Для досягнення максимальної продуктивності GPU NVIDIA CUDA необхідно обов'язково використовувати бібліотеки MAGMA або CUBLAS, які дають прискорення розглянутих розрахунків приблизно в 6.4 рази в порівнянні з використанням глобальної пам'яті (традиційний спосіб програмування).

6. Проведений аналіз стосується лише завдань матричного множення для заповнених несиметричних матриць. Невідомо, наскільки ефективно використання GPU для задач рішення систем лінійних рівнянь. Це є темою подальших досліджень.

Література

1. The ScaLAPACK Project. [Electronic resource]. - Mode of access: http://www.netlib.org, 2012

2. Automatically Tuned Linear Algebra Software (ATLAS). [Electronic resource]. - Mode of access: http://math-atlas.sourceforge.net/, 2012

3. Антонов А.С. Параллельное программирование с использованием технологии MPI: Учебное пособие. – М.: Изд-во МГУ, 2004. – 71 с.

4. Мясищев А.А. Достижение наибольшей производительности перемножения матриц на системах с многоядерными процессорами. - Кривий Ріг: Видавничий відділ НметАУ, 2010. – Т. 3: Теорія та методика навчання інформатики. – 303 с.

5. Гергель В.П. Теория и практика параллельных вычислений. [Electronic resource]. - Mode of access: http://www.intuit.ru/department/calculate/paralltp/lit.html.

6. Боресков А.В., Харламов А.А. Основы работы с технологией CUDA. - М.: ДМК Пресс, 2010. -232 с.: ил.

7. Matrix Algebra on GPU and Multicore Architectures. [Electronic resource]. - Mode of access: http://icl.cs.utk.edu/magma/index.html

8. CUBLAS (NVIDIA CUDA Basic Linear Algebra Subroutines). [Electronic resource]. - Mode of access: http://developer.nvidia.com/cublas

9. Установка CUDA Toolkit и драйвера NVIDIA для разработчиков. [Electronic resource]. - Mode of access: http://forum.ubuntu.ru/index.php?topic=114802.0

Надійшла до редакції
2.3.2012 р.

УДК 681.3

О.О. СЕМЕНОВА, А.О. СЕМЕНОВ
Вінницький національний технічний університет
А.В. РУДИК
Національний університет водного господарства та природокористування (м. Рівне)
В.В. ЧУХОВ
Житомирський державний технологічний університет

ВИКОРИСТАННЯ ПРИНЦИПІВ НЕЧІТКОЇ ЛОГІКИ ДЛЯ СИНТЕЗУ ЕЛЕМЕНТІВ ТРІЙКОВОЇ ЛОГІКИ

У статті пропонується використовувати широтно-імпульсні елементи нечіткої логіки у якості елементів трійкової логіки. Наведено схеми елементів доповнення/інверсії, мінімуму/кон'юнкції та максимуму/диз'юнкції. Описано функціонування вказаних елементів.

In this article we propose to use the pulse-width fuzzy logic elements as ternary and logic elements. The schemes of the complement/inversion, minimum/conjunction, and maximum/disjunction elements are presented. Operation of the elements is described.

Ключові слова: трійкова логіка, нечітка логіка, широтно-імпульсний.

Вступ

Одним з сучасних напрямків алгебри логіки є розроблення алгебри некласичних логік, таких як