УДК 519.683

д.т.н., проф. **Мясіщев О.А.** (ХмНУ)

# ЕФЕКТИВНІСТЬ ВИКОРИСТАННЯ GPU NVIDIA ПРИ ВИРІШЕННІ СИСТЕМ ЛІНІЙНИХ РІВНЯНЬ

***У статті проведені дослідження доцільності застосування графічних процесорів NVIDIA для розв'язання систем лінійних рівнянь в порівнянні зі звичайними багатоядерними процесорами. Показано, що GPU Tesla C2075 рішення систем лінійних рівнянь з матрицею загального вигляду виконує в 7 разів швидше для чисел з подвійною точністю, ніж паралельна обчислювальна система на базі шестиядерних процесора AMD Phenom II X6 1090T з частотою 3.2 ГГц.***

***Ключові слова: графічний процесор, факторизація матриці, GPU.***

***В статье проведены исследования целесообразности применения графических процессоров NVIDIA для решения систем линейных уравнений по сравнению с обычными многоядерными процессорами. Показано, что GPU Tesla C2075 решение систем линейных уравнений с матрицей общего вида выполняет в 7 раз быстрее для чисел с двойной точностью, чем параллельная вычислительная система на базе шестиядерного процессора AMD Phenom II X6 1090T з частотой 3.2 ГГц.***

***Ключевые слова: графический процессор, факторизация матрицы, GPU.***

***The paper investigated the feasibility of NVIDIA GPUs to solve systems of linear equations as compared to conventional multi-core processors. Shown that the GPU Tesla C2075 solving systems of linear equations with a general matrix performs 7 times faster for double-precision numbers than the parallel computer system with a six-core processor AMD Phenom II X6 1090T 3.2GHz h.***

***Keywords: GPU, matrix factorization, GPU.***

**Постановка проблеми.** Задачі теорії пружності, пластичності зводяться до системи диференціальних рівнянь в напругах і швидкостях. Зазвичай вони розв'язуються чисельно з використанням методу кінцевих елементів. У свою чергу цей метод зводиться до побудови так званих матриць жорсткості та рішенням систем лінійних рівнянь, що містить тисячі і навіть десятки тисяч невідомих. Для пружнопластичних і пластичних задач рішення систем рівнянь доводиться виконувати сотні і тисячі разів для виявлення кінцевої конфігурації деформованого матеріалу. Для цих цілей потрібні обчислювальні системи високої продуктивності. В даний час широкого поширення набули комп'ютери з багатоядерними процесорами. Наприклад, 6-й 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 близько 500 процесорних ядер, наприклад для Tesla C2075. Проблема полягає в тому, щоб распараллеліть рішення систем лінійних рівнянь 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 вирішення цього завдання 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 [1,2,3,6]. В роботі [4] представлені рішення ряду задач і на GPU NVIDIA. Однак не був проведений детальний аналіз можливостей сучасних GPU NVIDIA при використанні методів програмування на базі бібліотек програм MAGMA (Matrix Algebra on GPU and Multicore Architectures) [8] для розв'язання систем лінійних рівнянь з числами одинарної і подвійної точності.

**Мета статті.** В роботі досліджується доцільність застосування графічних процесорів при рішенні зокрема задач розв'язання систем лінійних рівнянь в порівнянні зі звичайними багатоядерними процесорами. Розглядаються особливості використання та проблеми установки бібліотеки MAGMA.

**Виклад основного матеріалу.** Для проведення обчислювальних експериментів розглянемо дві системи. У кожній з них встановлений процесор (CPU) AMD Phenom II X6 1090T з частотою 3.2 ГГц. У першій системі встановлений графічний процесор (GPU) Tesla C2075, у другій GeForce GTX 480 фірми NVIDIA. GPU будуть виконувати роль обчислювальних прискорювачів для розв'язання систем лінійних рівнянь. Причому в першому випадку розрахунок буде виконуватися с урахуванням розпаралелювання по 6-ядер процесора з використанням бібліотек MPI, ScaLAPACK і ATLAS [1,2,3]. У другому і третьому випадках - розпаралелюванням по ядрам GPU Tesla C2075 і GeForce GTX 480 з використанням технології CUDA [4]. Обчислювальні системи працюють під управлінням операційної системи Linux Ubuntu ver. 10.10 desktop (64bit). На них встановлені компілятори фортран F77, gfortran з перерахованими вище бібліотеки для 6-та ядерного процесора. Для програмування на GPU Tesla C2075 і GeForce GTX 480 інстальовані відеодрайвер nvidia і програмне забезпечення CUDA Toolkit 4.0 із сайту http://developer.nvidia.com/cuda-toolkit-archive. Послідовність установки програмного забезпечення для GPU представлена в джерелі [5].

У тестовій програмі на фортране розрахунок проводитиметься для чисел з подвійною точністю. Рішення системи рівнянь розбито на два етапи:

-факторизація матриці системи (A) [7];

-саме рішення системи з використанням результатів, отриманих на першому етапі.

Кожен етап виконується за допомогою звернення до відповідних підпрограм бібліотек ScaLAPACK, BLACS, ATLAS. Перший етап використовує підпрограму PDGETRF, другий - PDGETRS [2]. У програмі для кожної підпрограми розраховується час її виконання. Розраховуються кількість операцій для першого (ops) та другого етапу (ops1), і обчислюється продуктивність системи для LU-факторизації і рішення системи в цілому.

Розглянемо процедуру вирішення цієї ж задачі на GPU. Для забезпечення максимальної продуктивності будемо також використовувати готові бібліотеки для вирішення систем рівнянь методом LU - факторизації. З цією метою на обчислювальній системі були встановлені бібліотеки lapack-3.4.0 і MAGMA (Matrix Algebra on GPU and Multicore Architectures) версії 1.1.0 [8].

Нижче представлена для GPU програма ur_f1.f90, написана на fortran 90, для рішення системи рівнянь методом LU-факторизації:

```
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
```

Тут для LU-факторизації використовуються підпрограма magmaf_dgetrf_gpu, а для вирішення системи рівнянь - magmaf_dgetrs_gpu [8]. Для них також розраховується час виконання і продуктивності обчислень окремо для LU-факторизації та вирішенні системи в цілому.

Розглянемо результати обчислювальних експериментів. У таблиці 1 зведені результати розрахунків по представлених вище програмам для CPU AMD і GPU Tesla C2075 при подвійної точності обчислень. Для процесора даються два варіанти: розрахунок виконується одним ядром і 6-ма ядрами. Представлена величина прискорення, яка ніде не досягає шестиразовій величини. Результати наведені у вигляді дробу: чисельник - час рахунки в секундах, знаменник - продуктивність в гігафлопс в секунду. Окремими стовпцями представлені розрахунки для LU-факторизації і рішення системи. У стовпці "рішення" у чисельнику дано час роботи підпрограми PDGETRS, тобто зворотного ходу, а в знаменнику - сумарна продуктивність при рішенні системи. Видно, що основна трудомісткість обчислень

пов'язана з LU-факторизації. Для CPU вона в 26.491/0.121 = 219 разів більше часу зворотного ходу, а для GPU в 3.718/0.071 = 52 рази для розміру матриці 11008x11008. Співвідношення збільшується при збільшенні розміру рівнянь. Таким чином, оцінку продуктивності можна виконувати тільки по LU-факторизації.

Таблиця 1

| Мат-риця NxN | CPU AMD Phenom II X6 1090T | | | | | GPU Tesla C2075 | |
|---|---|---|---|---|---|---|---|
| | 6 ядер | | 1 ядро | | Прис-кор. | LU-фактор. | Рішен. |
| | LU-факт. | Рішен. | LU-факт. | Рішен. | | | |
| 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 |

У таблиці 2 наведені результати розрахунків продуктивностей виконання LU-факторизації в гігафлопс в секунду для GPU Tesla C2075, GeForce GTX 480 і CPU AMD Phenom II X6 1090T для одинарної (real) і подвійний (double) точність. Видно, що для розміру рівнянь до 3072, дешевший GPU GeForce 480 обганяє GPU Tesla C2075. При великих рівняннях перевага за Tesla C2075. В цілому при подвійної точності обчислень для розміру в 11008 рівнянь GPU Tesla C2075 “обганяє” процесор в 239.2/34.1 = 7.01 разів.

Таблиця 2

| Матриця 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 |

У таблиці 3 наведені результати обчислень першого аргументу системи рівнянь. Тут Xr (1) відповідає одинарної точності, а Xd (1) - подвійної точності. Видно, що чим менше розмір рівнянь, тим менше відрізняються Xr (1) і Xd (1). При розмірі системи в 3000 рівнянь можна "вірити" тільки першим трьом значущим цифрам першого аргументу, а при розмірі 10000 рівнянь похибка ставати такою великою, що й перші значущі цифри стають під питанням. Таким чином, для систем понад 3000 рівнянь доцільно використовувати при розрахунках тільки числа з подвійною точністю.

Таблиця 3

| Матриця NxN | Аргумент Xr(1), одинарна точн. | Аргумент Xd(1), подвійна точність | Різниця \|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 |

Рішення системи в 11008 рівнянь з числами подвійної точності для Xd (1) показали наступні результати:
GPU Tesla C2075
Xd(1)=**9.9392662447128E-004** CPU AMD
Phenom II X6=**9.9392662452686E-004**
Тут можна припустити, що перші дев'ять значущих цифр є правильними.

**Висновки та перспективи подальших досліджень.**

1. При вирішенні систем рівнянь оцінку продуктивності можна виконувати тільки по LU-факторизації.

2. Для систем понад 3000 рівнянь доцільно при розрахунках використовувати тільки числа з подвійною точністю.

3. При вирішенні систем до 3000 рівнянь доцільно використовувати більш дешевий GPU GeForce 480 для обчислень з будь-якою точністю. Він працює в цьому діапазоні рівнянь швидше CPU AMD Phenom II X6 1090T в 4 рази для подвійної точності і в 5.5 рази для одинарної.

4.Для системи в 11008 рівнянь при подвійної точності обчислень GPU "обганяє" CPU в 7 разів.

5. При вирішенні систем рівнянь на 6-й ядерному CPU вдається досягти максимального прискорення в 3.24 рази порівняно з розрахунками на 1-му ядрі для використовуваних бібліотек ScaLAPACK і ATLAS.

6. Перевага GPU в 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.

7. Розглянутий спосіб розв'язання систем лінійних рівнянь справедливий для використання в обчислювальній системі тільки одного GPU. Однак невідомо, наскільки ефективно розпаралелювання цього завдання на декількох встановлених GPU в багатоядерної системі.

ЛІТЕРАТУРА:

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

2. The ScaLAPACK Project. [Electronic resource]. – Mode of access: http://www.netlib.org

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

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

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

6.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.

7. LUP-разложение. Материал из Википедии – свободной энциклопедии. [Electronic resource]. – Mode of access: http://ru.wikipedia.org/wiki/LUP-разложение.

8. Matrix Algebra on GPU and Multicore Architectures. [Electronic resource]. – Mode of access: http://icl.cs.utk.edu/magma/