Компьютерная Математика

УДК 519.8

П.І. СТЕЦЮК, О.М. ХІМІЧ, В.А. СИДОРУК

РЕАЛІЗАЦІЯ r-АЛГОРИТМУ НА ГРАФІЧНИХ ПРОЦЕСОРАХ

Описано дві програмні реалізації r-алгоритму Шора з постійним коефіцієнтом розтягу простору та адаптивним регулюванням кроку. Перша реалізація виконана на мові octave, а друга – на мові Сі з використанням технології Nvidia CUDA. Проведено їх порівняльний аналіз для задачі мінімізації опуклої кусочно-лінійної функції.

Вступ. Прості матрично-векторні операції r-алгоритмів [1] роблять їх перспективними при реалізації методів розв'язання задач математичного програмування в системах паралельних або розподілених обчислень. Існуючі бібліотеки стандартних програм для паралельних матрично-векторних операцій дозволяють у короткі терміни адаптувати такі алгоритми для ефективної роботи з використанням векторних процесорів на основі графічних прискорювачів (GPU). Така реалізація дозволяє суттєво скоротити час обчислень за рахунок використання обчислювальних потужностей графічного процесора, які в рази перевищують обчислювальні потужності класичних процесорів.

У статті опишемо субградієнтний метод ralgb5, який є однією з реалізацій r-алгоритмів з адаптивним регулюванням кроку в напрямку нормованого антисубградієнта у перетвореному просторі змінних і постійним коефіцієнтом ~ для розтягу простору в напрямку різниці двох послідовних субградієнтів. Послідовність викладу матеріалу статті є наступною: 1) описано обчислювальну форму $r(\alpha)$-алгоритмів та її властивості; 2) описано $r(\alpha)$-алгоритми з адаптивним регулюванням кроку та їх особливості; 3) описана octave-функція ralgb5 та наведено її код; 4) для реалізації методу ralgb5 наведено код на мові Сі з використанням технології Nvidia CUDA та описано результати обчислювальних експериментів для опуклої кусочно-лінійної функції на графічному прискорювачі.

1. *r*(~)-алгоритм [2]. Нехай $f(x)$ – випукла функція, де x – вектор з n змінних. Мінімальне значення функції $f(x)$ будемо позначати $f^* = f(x^*)$, $x^* \in X^*$. Вважаємо, що $f(x)$ має обмежену множину мінімумів X^*, тобто виконується умова $\lim\limits_{\|x\|\to\infty} f(x) = +\infty$. Ця умова забезпечує коректність регулювання кроку в r -алгоритмах. Позначимо α – коефіцієнт розтягу простору, такий що $\alpha > 1$.

$r(\alpha)$-алгоритмом для мінімізації $f(x)$ називається ітеративна процедура знаходження послідовності n-мірних векторів $\{x_k\}_{k=0}^{\infty}$ і послідовності $n \times n$-матриць $\{B_k\}_{k=0}^{\infty}$ за наступним правилом:

$$x_{k+1} = x_k - h_k B_k \xi_k, \quad B_{k+1} = B_k R_\beta(\eta_k), \quad k = 0,1,2,\ldots, \tag{1}$$

де

$$\xi_k = \frac{B_k^T g_f(x_k)}{\| B_k^T g_f(x_k) \|}, \quad h_k \geq h_k^* = \arg\min_{h\geq 0} f(x_k - hB_k\xi_k), \tag{2}$$

$$\eta_k = \frac{B_k^T r_k}{\| B_k^T r_k \|}, \quad r_k = g_f(x_{k+1}) - g_f(x_k), \quad \beta = \frac{1}{\alpha} < 1. \tag{3}$$

Тут x_0 – стартова точка; $B_0 = I_n$ – одинична $n \times n$-матриця[1]; h_k^* – величина кроку з умови мінімуму функції $f(x)$ у напрямку нормованого антисубградієнта в перетвореному просторі змінних; $R_\beta(\eta) = I_n + (\beta - 1)\eta\eta^T$ – оператор стиснення простору субградієнтів у нормованому напрямку η з коефіцієнтом $\beta = \frac{1}{\alpha} < 1$; $g_f(x_k)$ і $g_f(x_{k+1})$ – субградієнти функції $f(x)$ у точках x_k та x_{k+1}. Якщо $g_f(x_k) = 0$, то $x_k = x^*$ і процес (1) – (3) зупиняється.

Коментар. На кожній ітерації r -алгоритмів реалізується субградієнтний спуск для опуклої функції $\varphi(y) = f(B_k y)$ у перетвореному просторі змінних $y = A_k x$, де $A_k = B_k^{-1}$. Насправді, якщо обидві частини формули $x_{k+1} = x_k - h_k B_k \xi_k$ помножити зліва на матрицю A_k, отримаємо

$$y_{k+1} = A_k x_{k+1} = A_k x_k - h_k \xi_k = y_k - h_k \frac{B_k^T g_f(x_k)}{\| B_k^T g_f(x_k) \|} = y_k - h_k \frac{g_\varphi(y_k)}{\| g_\varphi(y_k) \|}, \tag{4}$$

де вектор $g_\varphi(y_k) = B_k^T g_f(x_k)$ є субградієнтом функції $\varphi(y) = f(B_k y)$ у точці $y_k = A_k x_k$ простору змінних $y = A_k x$. Це легко бачити з того, що субградієнт функції $f(x)$ в точці x_k задовольняє нерівність

[1]Матрицю B_0 часто вибирають як діагональну матрицю D_n з додатними коефіцієнтами по діагоналі, за допомогою якої здійснюється масштабування змінних.

$$f(x) \geq f(x_k) + (g_f(x_k))^T (x - x_k) \quad \forall x \in E^n,$$

звідки, здійснюючи заміну змінних $x = B_k y$, отримуємо

$$\varphi(y) \geq \varphi(y_k) + (B_k^T g_f(x_k))^T (y - y_k) = \varphi(y_k) + (g_\varphi(y_k))^T (y - y_k) \quad \forall y \in E^n.$$

Якщо $h_k = h_k^*$, то формула (4) означає точний пошук мінімуму функції $\varphi(y) = f(B_k y)$ у напрямку нормованого антисубградієнта в перетвореному просторі змінних $y = A_k x$, а якщо $h_k \approx h_k^*$, то – наближений пошук. Якщо функція $f(x)$ є диференційованою у точці x_k, то можливий випадок $h_k = h_k^* = 0$, з яким пов'язані основні проблеми з умовою зупинки r-алгоритмів для негладких функцій.

Метод (1) – (3) називають B-формою r-алгоритмів; на кожній його ітерації коригується матриця, пов'язана із заміною змінних $x = By$. Ітерація методу вимагає порядку $5n^2$ арифметичних операцій множення, які визначають обчислювальну трудомісткість ітерації (операції додавання враховувати не будемо через їх малий вклад у трудомісткість ітерації). З них $3n^2$ множень потрібно для обчислення векторів $B_k \xi_k$, $B_k^T g_f(x_k)$ та $B_k^T r_k$ (множення матриці на вектор), а $2n^2$ множень вимагає одноранговий перерахунок матриці $B_{k+1} = B_k R_{\beta_k}(\eta_k)$. Дійсно,

$$B_{k+1} = B_k R_{\beta_k}(\eta_k) = B_k (I_n + (\beta_k - 1)\eta_k \eta_k^T) = B_k + (\beta_k - 1)(B_k \eta_k)\eta_k^T,$$

звідки легко бачити, що обчислення вектора $\eta = B_k \eta_k$ потребує n^2 множень і стільки ж множень вимагає побудова однорангової матриці $\eta\eta_k^T$.

2. *r*(~)-алгоритм з адаптивним кроком. Одним з ефективних зарекомендував себе $r(\alpha)$-алгоритм з адаптивним регулюванням кроку. Це варіант r-алгоритмів, в якому величина кроку h_k налаштовується в процесі виконання одновимірного спуску в напрямку нормованого антісубградієнта у перетвореному просторі змінних за допомогою параметрів h_0, q_1, n_h, q_2. Тут h_0 – величина початкового кроку (використовується на 1-*й* ітерації, на кожній наступній ітерації ця величина уточнюється); q_1 – коефіцієнт зменшення кроку ($q_1 \leq 1$), якщо умова завершення спуску в напрямку виконується за один крок; q_2 – коефіцієнт збільшення кроку ($q_2 \geq 1$); натуральне число n_h задає число кроків одновимірного спуску ($n_h > 1$), через кожні з яких крок буде збільшуватися в q_2 разів.

Умова завершення спуску в напрямку виконується як тільки виявлена точка x_{k+1}, для якої $(x_{k+1} - x_k)^T g_f(x_{k+1}) \geq 0$, і легко перевіряється, так як у силу позитивності кроку рівносильно виконання нерівності $(B_k B_k^T g_f(x_k))^T g_f(x_{k+1}) \leq 0$. Остання нерівність означає, що кут між двома

послідовними субградієнтами в перетвореному просторі змінних буде негострим. Дійсно, його можна записати як нерівність $(B_k^T g_f(x_k))^T B_k^T g_f(x_{k+1}) \le 0$, яка рівносильна нерівності $(g_\varphi(y_k))^T g_\varphi(y_{k+1}) \le 0$, де $g_\varphi(y_k) = B_k^T g_f(x_k)$ і $g_\varphi(y_{k+1}) = B_k^T g_f(x_{k+1})$ є субградієнтами функції $\varphi(y) = f(B_k y)$ у точках $y_k = A_k x_k$ и $y_{k+1} = A_k x_{k+1}$ перетвореного простору змінних $y = A_k x$, де $A_k = B_k^{-1}$. Оскільки передбачається, що $\lim\limits_{\|x\|\to\infty} f(x) = +\infty$, то після кінцевого числа кроків адаптивного спуску в напрямку нормованого антисубградієнта обов'язково виконається умова завершення спуску.

Ітеративний процес в $r(\sim)$-алгоритмі з адаптивним регулюванням кроку триває до виконання деякого критерію зупинки, де ключову роль грають параметри ε_x і ε_g. Вони визначають дві умови зупинки: метод зупиняється у точці x_{k+1}, якщо виконана умова $\| x_{k+1} - x_k \| \le \varepsilon_x$ (зупинка за аргументом); метод зупиняється у точці x_{k+1}, якщо виконана умова $\| g_f(x_{k+1}) \| \le \varepsilon_g$ (зупинка за нормою субградієнта, використовується для гладких функцій). Крім них використовуються ще дві умови зупинки методу: стандартна зупинка, якщо перевищено максимальну кількість ітерацій, і аварійна зупинка, яка сигналізує про те, що або функція $f(x)$ необмежена знизу, або початковий крок h_0 занадто малий і його потрібно збільшити. І хоча $r(\alpha)$-алгоритм з адаптивним регулюванням кроку не гарантує монотонного спадання функції, однак, як показали експерименти, зростання функції відбувається досить рідко. Докладні рекомендації за вибором коефіцієнта розтягу простору і параметрів адаптивного регулювання кроку можна знайти в [3, с. 45 – 47]. Суть їх вибору полягає у тому, щоб адаптивний спосіб регулювання кроку дозволяв збільшувати точність пошуку мінімуму функції у напрямку в процесі рахунку і при цьому число кроків у напрямку не повинно перевищувати в середньому двох-трьох на одну ітерацію.

Якщо $r(\alpha)$-алгоритм з адаптивним регулюванням кроку застосовувати для мінімізації негладких функцій, то рекомендується наступний вибір параметрів: $\alpha = 2 \div 4$, $h_0 = 1.0$, $q_1 = 1.0$, $q_2 = 1.1 \div 1.2$, $n_h = 2 \div 3$. Якщо відома апріорна оцінка відстані від початкової точки x_0 до точки мінімуму x^*, то початковий крок h_0 доцільно вибирати порядку $\| x_0 - x^* \|$. При мінімізації негладких функцій рекомендовані параметри такі ж, за винятком q_1 ($q_1 = 0.8 \div 0.95$). Це обумовлено тим, що додаткове подрібнення кроку сприяє підвищенню точності пошуку мінімуму функції у напрямку, що при мінімізації гладких функцій забезпечує більш швидку швидкість збіжності. За такого вибору параметрів, як правило, число спусків у напрямку рідко перевищує два, а за n кроків точність за функцією поліпшується в три – п'ять разів. Параметри

зупинки $\varepsilon_x, \varepsilon_g \sim 10^{-6} \div 10^{-5}$ при мінімізації опуклої функції, навіть істотно яружної структури, забезпечують знаходження точки x_r^* – наближення до точки x^*, для якого значення функції, досить близьке до оптимального: $\frac{f(x_r^*) - f^*}{|f^*|+1} \sim 10^{-6} \div 10^{-5}$ – для негладких і $\frac{f(x_r^*) - f^*}{|f^*|+1} \sim 10^{-12} \div 10^{-10}$ – для гладких функцій. Це підтверджується результатами численних тестових і реальних розрахунків.

3. Octave-функція ralgb5 [4, с. 383 – 386]. Програма **ralgb5** знаходить точку x_r^* – наближення до точки мінімуму опуклої функції $f(x)$ від *n* змінних, яка для $r(\alpha)$-алгоритму з адаптивним кроком визначається вибраними: стартовою точкою x_0; коефіцієнтом розтягу простору α, параметрами адаптивного регулювання кроку h_0, n_h, q_1 та q_2; параметрами зупинки ε_g, ε_x та **maxitn**. Програма використовує octave-функцію вигляду **function [f,g] = calcfg(x)**, яка обчислює значення функції $f = f(x)$ і її субградієнта $g = g_f(x)$ у точці x. Ім'я функції вигляду **calcfg(x)** може бути довільним, яке допускає синтаксис мови octave. Програма **ralgb5** використовує такі вхідні та вихідні параметри.

```
%  Вхідні параметри:
%   calcfg -- ім'я функції виду calcfg(x) для обчислення f i g
%    x  -- початкова точка x0(1:n) (на виході псується)
%    alpha -- коефіцієнт розтягу простору
%    h0,nh,q1,q2 -- параметри адаптивного регулювання кроку
%    epsx,epsg,maxitn -- параметри зупинки
%   Вихідні параметри:
%    xr -- знайдена точка мінімуму функції xr(1:n)
%    fr -- значення функції f в точці xr
%    itn – кількість виконаних ітерацій
%    ncalls – кількість викликів функції calcfg
%   istop -- код зупинки  (2 = epsg, 3 = epsx, 4 = maxitn, 5 = error)
```

За точку x_r приймається така точка ітераційного процесу, що реалізує $r(\alpha)$-алгоритм з адаптивним кроком, в якій отримано найменше (рекордне) значення функції $f_r = f(x_r)$. Вона не обов'язково збігатиметься з останньою точкою ітераційного процесу, а може бути отримана при одновимірному пошуку мінімуму функції у напрямку на будь-якій з попередніх ітерацій процесу. Статус точки x_r визначається кодом зупинки istop на ітерації itn = *k:*

1) зупинка сталася за умови $\| g_f(\tilde{x}_k) \| \le \varepsilon_g$, де $\tilde{x}_k \in [x_k, x_{k+1}]$ (тут $x_r = \tilde{x}_k$ і за малих ε_g функція $f(x)$ диференційована в рекордній точці x_r);

2) зупинка сталася за умови $\| x_{k+1} - x_k \| \le \varepsilon_x$ ($x_r = x_{k+1}$, якщо $f(x_{k+1})$ менше, ніж значення функції в інших точках ітераційного процесу);

3) зупинка сталася за умови itn>maxitn (перевищено максимальну кількість ітерацій і близькість точки x_r до оптимальної x^* можна оцінити за відхиленнями f_r від значень функції на останніх операціях);

4) зупинка сталася через те, що за 500 кроків у напрямку не виконана умова завершення спуску (ця зупинка вважається аварійною).

Аварійна зупинка пов'язана або з тим, що функція $f(x)$ необмежена знизу, або початковий крок h_0 занадто малий і його потрібно збільшити. Це може бути як через те, що для опуклої функції $f(x)$ не виконується умова $\lim\limits_{\|x\|\to\infty} f(x) = +\infty$, так і через помилки в програмі обчислення значення функції та субградієнта. Помилка при обчисленні функції менш небезпечна (вона впливає тільки на визначення рекордної точки x_r), ніж помилка при обчисленні компонент субградієнта (градієнта), який визначає вибір напрямку одновимірного спуску. Аварійна зупинка через малу величину кроку малоймовірна, тому що **500** кроків для спуску в напрямку вибрано з тих міркувань, щоб для рекомендованого мінімального значення $q_2 = 1.1$ початковий крок для чергової ітерації міг максимально збільшитись у 10^6 разів, якщо $n_h = 3$, та в 10^{10} разів, якщо $n_h = 2$.

Програма **ralgb5** оформлена як octave-функція, код її є наступним:

```
# Octave-function ralgb5 for Shor's r-algorithm
function [xr,fr,itn,ncalls,istop] = ralgb5(calcfg,x,alpha,h0,q1, # row001
                                           q2,nh,epsg,epsx,maxitn);
itn = 0; hs = h0; B = eye(length(x)); xr = x;                    # row002
ncalls = 1; [fr,g0] = calcfg(xr);                                # row003
printf("itn %4d f %14.6e fr %14.6e ls %2d ncalls %4d\n",         # row004
       itn, fr, fr, 0, ncalls);
if(norm(g0) < epsg) istop = 2;  return; endif                    # row005
for (itn = 1:maxitn)                                             # row006
   dx = B * (g1 = B' * g0)/norm(g1);                             # row007
   d = 1; ls = 0; ddx = 0;                                       # row008
   while (d > 0)                                                 # row009
      x -= hs * dx; ddx += hs * norm(dx);                        # row010
      ncalls ++; [f, g1] = calcfg(x);                            # row011
      if (f < fr)  fr = f; xr = x; endif                         # row012
      if(norm(g1) < epsg) istop = 2;  return; endif              # row013
      ls ++; (mod(ls,nh)==0) && (hs *= q2);                      # row014
      if(ls > 500) istop = 5;  return; endif                     # row015
      d = dx' * g1;                                              # row016
   endwhile                                                      # row017
   (ls == 1) && (hs *= q1);                                      # row018
   printf("itn %4d f %14.6e fr %14.6e ls %2d ncalls %4d\n",      # row019
          itn, f, fr, ls, ncalls);
   if(ddx < epsx) istop = 3;  return; endif                      # row020
   xi = (dg = B' * (g1 - g0) )/norm(dg);                         # row021
   B += (1 / alpha - 1) * B * xi * xi';                          # row022
   g0 = g1;                                                      # row023
endfor                                                           # row024
istop = 4;                                                       # row025
endfunction                                                      # row026
```

Наведений код складається з **26** рядків і велика частина з них містить більше одного оператора мови **octave**. Функціонально він повністю збігається з кодом програми **ralgb5**, який наведено в [4, с. 384 – 385], та відрізняється від нього тільки нумерацією рядків і незначним переформатуванням тексту.

Код octave-функції **ralgb5** реалізований за формулами методу (1) – (3) так, що ітераційний процес виконується в циклі **for** (рядки **6 – 24**), де для *k*-ої ітерації субградієнт $g_f(x_k)$ запам'ятовується вектор-стовпцем **g0**, а субградієнт $g_f(x_{k+1})$ запам'ятовується вектор-стовпцем **g1**. Адаптивний спосіб регулювання кроку на ітерації реалізується рядками **8 – 18** на основі внутрішнього циклу **while**, який закінчується тоді, як тільки виконується умова завершення спуску в напрямку, обчисленому в рядку **7**. У циклі **while** оновлюються «рекордні» точка та значення функції (рядок **12**) і перевіряються дві умови зупинки: зупинка за нормою градієнта ε_g (рядок **13**) та аварійна зупинка (рядок **14**). Рядок **19** забезпечує друк на кожній ітерації її номера **itn**, значення функції **f**, рекордного значення функції **fr**, кількості кроків одновимірного спуску на ітерації **ls** і загальної кількості викликів програми для обчислення значень функції і її субградієнта **ncalls**. В рядку **20** реалізується умова зупинки за аргументом, в рядках **21 – 23** реалізуються перерахунок матриці B та установка субградієнта **g0** для переходу до наступної ітерації. У рядках **2 – 5** ініціалізуються величини, необхідні для старту основного циклу **for** за ітераціями (рядок **6**).

За допомогою програми **ralgb5** можна знаходити досить точні наближення до точки мінімуму опуклої функції. Якщо коефіцієнт розтягу простору вибрати таким, щоб він добре узгоджувався з параметрами адаптивного регулювання кроку в напрямку нормованого антисубградієнта в перетвореному просторі змінних, то для виконання одних і тих же критеріїв зупинки можна значно скоротити кількість ітерацій і кількість обчислень значення функції і субградієнта (градієнта). Це залежить від конкретного виду функції, що мінімізується, ступеня її яружності та масштабу змінних.

4. Реалізація гібридного алгоритму. Програмний код процедури ralgb5 легко можна реалізувати засобами мови Сі з використанням технології Nvidia CUDA [5]. Така реалізація дозволить суттєво скоротити час обчислень за рахунок використання обчислювальних потужностей графічного процесора, які в рази перевищують обчислювальні потужності класичних процесорів. Далі приведено код, який відповідає за реалізацію алгоритму ralgb5 на графічному прискорювачі.

```
eye << <1, n >> > (B, n);
sabs<<<1,n>>>(xr, n, fr, g0, q);
printf("itn %4d f %16.8e fr, %21.13e ls %2d ncalls %4d\n", itn, fr, fr, 0,
ncalls);
norm = cublasDnrm2(n, g0, inc);
if (norm < epsg) {istop = 2;return istop;}
for (itn = 1; itn < maxitn; itn++)
{
double d = 1; int ls = 0; double ddx = 0;
```

```
cublasDgemv('T', n, n, alpha, B, n, g0, inc, beta, g1, inc);
norm = cublasDnrm2(n, g1, inc);
cublasDgemv('N', n, n, alpha, B, n, g1, inc, beta, dx, inc);
cublasDscal(n, 1 / norm, dx, inc);
while (d>0)
{
       kernel << <1, n >> >(x, dx, hs);
       norm1 = cublasDnrm2(n, dx, inc);
       ddx += hs*norm1; ncalls++;
       sabs<<<1,n>>>(x, n, f, g1, q);
       if(f<fr){fr=f; cudaMemcpy(xr,x,n*sizeof(double),cudaMemcpyDefault);}
       if (norm<epsg){ istop = 2; break; }
       ls++;
       if ((ls % (int)nh) == 0){ hs *= q2; }
       if (ls>500){istop = 5; break; }
       d = cublasDdot(n, dx, inc, g1, inc);
}
if (ls == 1){hs *= q1; }
printf("itn %4d f %16.8e fr %21.13e ls %2d ncalls %4d\n", itn, f, fr, ls,
ncalls);
if (ddx<epsx){ istop = 3; break; }
sum_m_m_po_elem << <1, n >> >(g1, g0, res, -1);
cublasDgemv('T', n, n, alpha, B, n, res, inc, beta, dg, inc);
norm = cublasDnrm2(n, dg, inc);
cublasDscal(n, 1 / norm, dg, inc);
cublasDgemv('N', n, n, gama, B, n, dg, inc, beta, xi1, inc);
cublasDgemm('N', 'T', n, n, inc, alpha, xi1, n, dg, n, alpha, B, n);
cudaMemcpy(g1, g0, n*sizeof(double), cudaMemcpyDefault);
}
istop = 4;
```

Варто зазначити, що в коді використовуються функції бібліотеки CUBLAS [6] (ефективна реалізація BLAS функцій на GPU) та оригінальні програми на мові CUDA.

Проведено ряд розрахунків для задач мінімізації опуклої кусочно-лінійної функції $f(x_1,\ldots,x_n)=\sum_{i=1}^{n} q^{(i-1)}\left|x_i-1\right|$ з використанням реалізацій r-алгоритму на мовах Octave Сі. Для обчислення значення функції $f(x_1,\ldots,x_n)=$ $=\sum_{i=1}^{n} q^{(i-1)}\left|x_i-1\right|$ та її субградієнта для мови Octave використовувався наступний код на мові Octave:

```
function [f,g] = sabs(x)
global q;
n=length(x);
f=0.d0; a1=1.d0;
for (i = 1:n)
   f=f+a1*abs(x(i)-1.d0);
   if (x(i)-1.0>=0.) g(i) = a1; endif
   if (x(i)-1.0<0.)  g(i) = -a1; endif
   a1=a1*q;
endfor
g = g';
endfunction
```

Відповідна реалізація на мові CUDA має наступний вигляд.

```
__global__ void sabs(double* in, int n, double out1, double* out2, double q2)
{
	int i = blockIdx.x*blockDim.x + threadIdx.x;
	double a1 = 1.0; out1 = 0;
	if (i < n)
	{
		out1 = out1 + a1*abs(in[i] - 1.0);
		if (in[i] - 1 >= 0) out2[i] = a1;
		if (in[i] - 1 < 0) out2[i] = -a1;
		a1 = a1*q2;
	}
};
```

В таблиці далі приводяться наступні дані: *n* – кількість змінних у кусочно-лінійній функції, t_octave – час розрахунку з використанням octave-функції, t_hybrid – час розрахунку з використанням гібридної реалізації алгоритму.

З таблиці видно, що для систем порядку більше 500 гібридна реалізація алгоритму є кращою. Скорочення часу, яке досягається, варіюється від 14 до 18 разів. Параметри системи на якій проводились розрахунки наступні: процесор Athlon II x4 641 2.8 Ghz (4 cores), RAM – 8 Gb DDR3 1066, GPU – Nvidia GeForce GTS 450 1Gb (96 Cuda cores). Версія Octave: Octave 4.0.2. Версія CUDA: 7.5.

ТАБЛИЦЯ. Порівняння реалізацій *r*-алгоритму на мовах Octave та Сі

n	t_octave (сек.)	t_hybrid (сек.)	Speedup
500	4.9993	0.293333	17.04
1000	11.831	0.821053	14.41
2000	34.593	1.9	18.2
4000	112.11	6.840625	16.38
8000	391.06	22.36581	17.48

Висновки. Субградієнтний метод ralgb5 можна використовувати при вирішенні негладких задач з різних областей застосувань. Так як гладка функція з дуже швидко мінливим градієнтом за своїми властивостями близька до негладкої функції, то він має прискорену збіжність при оптимізації яружних гладких функцій. Octave-функцію ralgb5 можна використовувати як оптимізаційне ядро при реалізації на мові octave алгоритмів розв’язання задач нелінійного програмування. Суттєве прискорення можна отримати, використовуючи для розрахунків технологію CUDA на графічних прискорювачах.

Робота виконана за підтримки НАН України (проекти № 0116U004558, № 0116U006078).

П.И. Стецюк, А.Н. Химич, В.А. Сидорук

РЕАЛИЗАЦИЯ r-АЛГОРИТМА НА ГРАФИЧЕСКИХ ПРОЦЕССОРАХ

Описаны две программные реализации r-алгоритма Шора с постоянным коэффициентом растяжения пространства и адаптивным регулированием шага. Первая реализация выполнена на языке octave, вторая – на языке Си с использованием технологии Nvidia CUDA. Проведен их сравнительный анализ для задачи минимизации выпуклой кусочно-линейной функции.

P.I. Stetsyuk, A.N. Khimich, V.A. Sydoruk

AN IMPLEMENTATION OF r-ALGORITHM ON GPUs

We describe two software implementations of Shor's r-algorithm with constant coefficient of space dilation and step adaptive control. The first implementation is developed in octave language and the second is in C language using Nvidia CUDA technology. A comparative analysis for the problem of convex piecewise-linear function minimizing is conducted.

1. *Шор Н.З.* Методы минимизации недифференцируемых функций и их приложения. Киев. Наук. думка, 1989. 200 с.
2. *Сергиенко И.В., Стецюк П.И.* О трех научных идеях Н.З. Шора. *Кибернетика и системный анализ.* 2012. № 1. С. 4 – 22.
3. *Шор Н.З., Стеценко С.И.* Квадратичные экстремальные задачи и недифференцируемая оптимизация. Киев: Наук. думка, 1989. 208 с.
4. *Стецюк П.И.* Методы эллипсоидов и r-алгоритмы. Кишинэу, Эврика, 2014. 488 с.
5. CUDA C Programming Guide Version 4.2. – Santa Clara: Nvidia, 2012. 173 p.
6. https://developer.nvidia.com/cuBLAS

Одержано 15.11.2016

Про авторів:

Стецюк Петро Іванович,
доктор фізико-математичних наук,
завідувач відділу
Інституту кібернетики імені В.М. Глушкова НАН України,
E-mail: stetsyukp@gmail.com

Хіміч Олександр Миколайович,
член-кореспондент НАН України, доктор фізико-математичних наук,
завідувач відділу
Інституту кібернетики імені В.М. Глушкова НАН України,
E-mail: khimich505@gmail.com

Сидорук Володимир Антонович,
кандидат фізико-математичних наук,
молодший науковий співробітник
Інституту кібернетики імені В.М. Глушкова НАН України.
E-mail: wolodymyr.sydoruk@gmail.com