Компьютерная Математика

УДК 519.6

О.В. ЧИСТЯКОВ

ПРО ОСОБЛИВОСТІ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗВ'ЯЗАННЯ ЗАДАЧ НА ВЛАСНІ ЗНАЧЕННЯ З РОЗРІДЖЕНИМИ МАТРИЦЯМИ НА ГІБРИДНИХ КОМП'ЮТЕРАХ

Розглядаються проблеми розробки паралельного алгоритмічно-програмного забезпечення для розв'язування задач на власні значення на комп'ютерах гібридної архітектури. Запропоновано деякі шляхи підвищення ефективності паралельних алгоритмів на прикладі реалізації гібридних ітераційних алгоритмів для розв'язання часткової алгебраїчної проблеми власних значень для симетричних додатно-визначених матриць розрідженої структури.

Вступ. При дискретизації задач математичної фізики дуже часто виникають задачі розв'язання часткової алгебраїчної проблеми власних значень з симетричними додатно-визначеними розрідженими матрицями, розміри яких можуть досягати десятки мільйонів. Розв'язування таких задач потребує значних обчислювальних ресурсів, які можуть бути надані сучасними гібридними комп'ютерами що поєднують обчислення на багатоядерних комп'ютерах MIMD-архітектури з прискоренням обчислень на графічних процесорах SIMD-архітектури. Ефективність реалізації паралельних обчислень у великій мірі залежить від того, наскільки характерні властивості задачі та алгоритму для її розв'язування відповідають особливостям гібридного комп'ютера та систем розпаралелення. В роботі проводиться дослідження особливостей гібридних комп'ютерів, які впливають на ефективність паралельних ітераційних алгоритмів знаходження найменшого власного значення та відповідного власного вектора симетричної додатно-визначеної блочно-діагональної матриці з обрамленням на прикладі застосування однокрокового паралельного алгоритму на основі попереміннотрикутного методу [1].

Архітектурні особливості CPU та системи розпаралелення MPI. Реалізація обчислень на гібридному комп'ютері передбачає використання двох різних архітектур:

MIMD-архітектури − багатоядерних процесорів CPU та графічних процесорів (GPU). Кожна з цих архітектур дає можливість здійснювати різного рівня розпаралелення, проте без урахування особливостей архітектури гібридної системи та технологій програмування неможливо забезпечити високу швидкодію алгоритмів та програм. Розглянемо коротко деякі суттєві відмінності між центральним процесором та відеокартою.

MIMD-комп'ютер, як правило, складається з обчислювальних вузлів, кожен з яких може мати декілька багатоядерних процесорів та свою локальну пам'ять, яка розподіляється між цими процесорами, тобто кожен процесор має доступ тільки до локальної пам'яті свого вузла. Обчислювальні вузли об'єднуються між собою деяким комунікаційним середовищем, тому доступ до даних, які розміщені в пам'яті інших вузлів, виконується довше. Програми, які реалізують розв'язування задач на CPU, можуть звертатися безпосередньо до будь-яких осередків лінійної і однорідної пам'яті. Крім того, процесори мають внутрішні кеші інструкцій та даних, досить великих розмірів. Швидкодію обчислювальних алгоритмів та програм на CPU можна значно підвищити, якщо розробляти їх з урахуванням наявної кеш-пам'яті процесора. Очевидно, що матрично-векторні операції будуть виконуватися найбільш ефективно лише для тих порядків матриць, які будуть повністю поміщатися в кеш-пам'ять процессора CPU.

Для розпаралелення алгоритмів на CPU використовується система MPI [2], функції якої дають можливість розподіляти обчислення між ядрами/вузлами CPU у вигляді процесів, що виконуються паралельно та обмінюються між собою інформацією, забезпечувати синхронізацію обчислень.

Архітектурні особливості GPU та технології розпаралелення CUDA. CUDA [3] − це програмно-апаратна обчислювальна архітектура NVIDIA, заснована на розширенні мови *C*, за допомогою якої здійснюється доступ до набору інструкцій графічного прискорювача і управління його пам'яттю при організації паралельних обчислень. CUDA дозволяє писати спеціальні програмні функції − ядра (*kernels*), які виконуються паралельно на GPU у вигляді безлічі різних потоків (*threads*).

На GPU є 6 видів пам'яті, кожна з яких має своє призначення і до яких можна звертатися. Ми зупинимося більш детально на двох видах пам'яті, які мають найбільший вплив на ефективність виконання задач.

Глобальна пам'ять (*global)* необхідна в основному для відправлення даних задачі з хоста (CPU) на GPU та для збереження результатів роботи програми перед відправленням їх на хост. Вона працює дуже повільно, тому кількість звернень до глобальної пам'яті слід у будь-якому випадку мінімізувати. Копіювання даних з пам'яті CPU в пам'ять GPU і назад відноситься до найбільших обчислювальних витрат та суттєво збільшують час виконання програми. В цьому випадку доцільно передбачити синхронне виконання копіювання даних від CPU до GPU та обчислень на CPU.

Розділена пам'ять (*shared*-пам'ять) – це швидка пам'ять. Саме цю пам'ять рекомендується використовувати як керований кеш при розв'язуванні задач.

Проте ця пам'ять дуже мала. На один мультипроцесор доступно всього 16 Кбайт розподіленої пам'яті. Блоки, які виявилися найбільш придатними для кеш-пам'яті CPU, не помістяться в ній. Тому доцільніше спочатку розбити матриці на смуги по 16 стовпчиків, скопіювати ці смуги на глобальну пам'ять GPU. Це достатньо велика пам'ять для збереження інформації. Розбивши смуги матриць на підматриці 16×16, можна організувати матрично-векторні операції цих підматриць на *shared*-пам'яті.

Реалізація паралельного поперемінно-трикутного алгоритму на гібридному комп'ютері. Розглянемо задачу на власні значення

$$Ax = \lambda Dx, \tag{1}$$

де A, D – розріджені додатно-визначені матриці порядку n, що діють в n-вимірному евклідовому просторі H зі скалярним добутком $(\ ,\ )$, λ та x – відповідне власне значення та власний вектор.

Застосуємо до задачі (1) наступну канонічну ітераційну однокрокову схему знаходження λ_1 та x_1:

$$B(y_{k+1} - y_k) + \tau_{k+1} r_k = 0, \ \text{для } k = 0,1,2,3,\ldots,$$

де y_0 – довільне початкове значення, $r_k = Ay_k - \mu_k Dy_k$ – нев'язка; $\mu_k = (Ay_k, y_k)(y_k, y_k)^{-1}$ – наближення до власного значення; y_k – нормоване наближення до власного вектора, τ_k – ітераційний параметр, B – оператор (регуляризатор), що покращує швидкість ітераційного процесу та для якого легко знаходиться обернений.

Розлянемо паралельний ітераційний алгоритм для симетричних додатно-визначених матриць, попередньо приведених до блочно-діагональних матриць з обрамленням, і використання B, як трикутного факторизованого оператора, а саме поперемінно трикутний алгоритм.

Для вибору передобумовлювача застосуємо до вихідної матриці метод паралельних перерізів, який приводить вихідну матрицю до блочно-діагонального вигляду з обрамленням:

$$\hat{A} = P^T AP = \begin{pmatrix} D_1 & 0 & 0 & \ldots & 0 & B_1 \\ 0 & D_2 & 0 & \ldots & 0 & B_2 \\ 0 & 0 & D_3 & & 0 & B_3 \\ \vdots & \vdots & & \ddots & & \vdots \\ 0 & 0 & 0 & & D_{p-1} & B_{p-1} \\ B_1 & B_2 & B_3 & \ldots & B_{p-1} & D_p \end{pmatrix},$$

де P – матриця перестановок, а блоки D_i і B_i зберігають розрідженість.

Таким чином, початкова задача зводиться до наступної:

$$\hat{A}y = \lambda Dy, \qquad x = Py.$$

Оскільки використано перетворення подібності, то власні значення отриманої матриці і вихідної збігаються.

Для попереміннo-трикутного методу передобумовлювач має вигляд:

$$B = (E + w\hat{R})(E + w\hat{R}^T), \quad \hat{A} = \hat{R} + \hat{R}^T.$$

При цьому $\hat{R}$ зберігає блочно-трикутну структуру, успадковану від матриці $\hat{A}$:

$$\hat{R} = \begin{pmatrix} \hat{R}_1 \\ \hat{R}_2 \\ \hat{R}_3 \\ \vdots \\ \hat{R}_{p-1} \\ \hat{R}p \end{pmatrix} = \begin{pmatrix} \tilde{D}_1 & 0 & 0 & \dots & 0 & 0 \\ 0 & \tilde{D}_2 & 0 & \dots & 0 & 0 \\ 0 & 0 & \tilde{D}_3 & \dots & 0 & 0 \\ \vdots & \vdots & & \ddots & & \dots \\ 0 & 0 & 0 & & \tilde{D}_{p-1} & 0 \\ C_1 & C_2 & C_3 & \dots & C_{p-1} & \tilde{D}_p \end{pmatrix}, \tag{2}$$

де блоки $\tilde{D}_i$ $(1 \le i \le p)$ – нижні трикутні матриці блоків D_i.

Розглянемо основні обчислювальні процедури попереміннo-трикутного методу при наступному розподілі блоків (підматриць) $\hat{R}$ по процесах CPU. З огляду на структуру $\hat{R}$ це означає, що процеси з номерами $0 \le i < p$ зберігають блоки $\tilde{D}_i$ та C_i, а процес з номером $p - 1$ зберігає блок $\tilde{D}_p$. Тут p – загальна кількість процесів.

Реалізація алгоритму визначається блочно-трикутною структурою матриць $\hat{R}_i$ при розв'язанні системи $Bw = r$, $B = (E + \hat{\omega}R)(E + \hat{\omega}R^T)$, $\hat{A} = \hat{R} + \hat{R}^T$.

Паралельний алгоритм розв'язання нижньої трикутної системи $(E + \omega\hat{R})y = r$ зводиться до одночасного та незалежного виконання на окремих процесах розв'язування трикутних систем $(E + \omega\tilde{D}_q)y_q = r_q$, $1 \le q < p$, та наступного обчислення $\tilde{y}_q = C_q y_q$, $0 \le q < p$, де q – номер GPU, після чого останній процес сумує значення $\tilde{y}_q$, надіслані від інших GPU і знаходить y_p, розв'язуючи систему $(E + \omega\hat{D}p)y_p = r_p - \sum_{q=1}^{p-1} y_q$.

Аналогічно для розв'язку системи $(E + \omega R^T)w = y$ p-й процес розв'язує систему $(E + \omega\tilde{D}_p^{\mathrm{T}})w_p = y_p$ і розсилає компоненти w_p іншим процесам, вони незалежно розв'язують системи: $(E + \omega\tilde{D}_q^{\mathrm{T}})w_q = y_q - C_q^{\mathrm{T}} w_p$.

Ефективність ітераційних алгоритмів для розв'язування задач з розрідженими матрицями на гібридному комп'ютері в значній мірі залежить від накладних витрат при виконанні матрично-векторних операцій.

Однією з найбільш складних за ефективністю виконання є операція множення розрідженої матриці на вектор. При виконанні цієї операції на процесах виникають труднощі, які пов'язані з великою кількістю накладних витрат, наприклад, з індексацією вхідних даних та обмінами між процесами.

Крім перемноження матриці на вектор, в алгоритмі присутні ще два типи базових операцій. Один з них – додавання векторів та множення вектора на число. Виходячи з блочної схеми розбиття даних між обчислювальними процесами, кожен з них містить певні частини векторів. Отже, зазначені операції можуть бути виконані локально без будь-яких комунікацій з сусідніми вузлами.

Для підвищення ефективності виконання матрично-векторних операцій доцільно застосовувати бібліотеку програм обчислювальної математики Intel MKL [4] на CPU та бібліотеки матричних операцій на GPU CuBLAS, CuSparse [5].

Формати збереження розріджених матриць. При створенні алгоритмів розв'язування задач з розрідженими матрицями велике значення має вибір способів задання та збереження ненульових елементів. Ці способи визначаються насамперед структурою (розміщенням ненульових елементів) розрідженої матриці, вимогами алгоритму, який використовується, а також архітектурою гібридного комп'ютера.

Складність ефективної обробки розріджених матриць привела до створення великої кількості форматів їх збереження в комп'ютері, наприклад, CSR (compressed sparse row), CSC (compressed sparse column), COO (coordinate) та інші. Не можна рекомендувати єдиний, близький до оптимального, формат зберігання даних. Це у великій мірі залежить від виду розрідженої матриці. В працях багатьох авторів [6] проведено аналіз різних форматів збереження розріджених матриць, який показав, що формат CSR, який належить до найбільш універсальних і дозволяє легко реалізувати оптимальний алгоритм множення розрідженої матриці на вектор на CPU, не завжди забезпечує таку продуктивність для цієї операції на GPU. У ряді випадків виявляється, що переваги мають модифіковані формати, які враховують розрідженість матриці, архітектурні особливості гібридного комп'ютера та організації пам'яті на GPU.

Оскільки розріджені матриці великої розмірності потребують великих обсягів обчислювальних ресурсів для їх збереження в комп'ютері та витрат часу на їх обробку, виникає потреба у виконанні великої кількості додаткових операцій при їх переформатуванні або переіндексації для забезпечення масштабованості обчислювального процесу. Тому був розроблений комбінований формат збереження матриць, що базується на CSR форматі та враховує особливості представлення блочно-діагональної розрідженої матриці з обрамленням. Схематично комбінований формат блочно-діагональної розрідженої матриці з обрамленням показано на рис. 1.

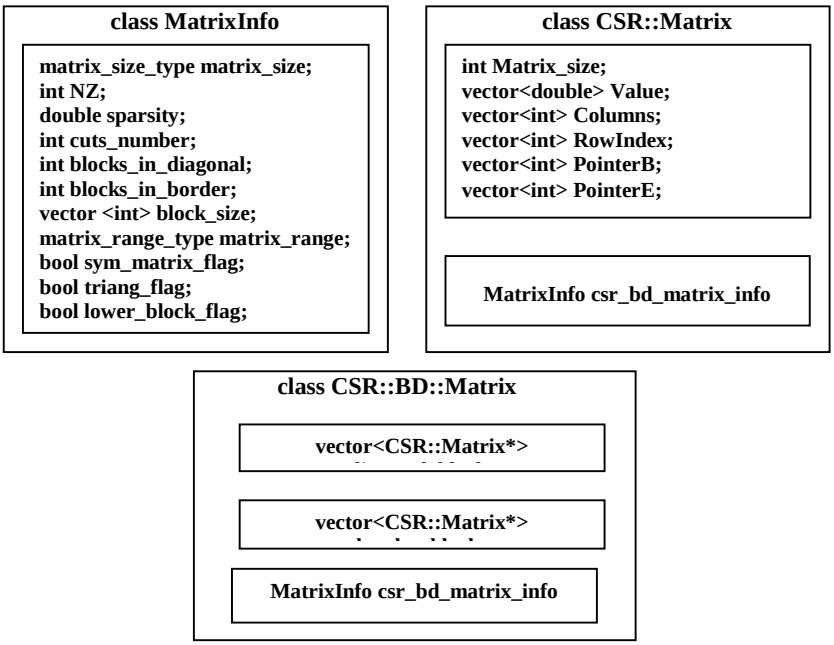

РИС. 1. Комбінований формат збереження блочно-діагональної розрідженої матриці з обрамленням

Зі схеми на рис. 1 видно, що блочно-діагональна матриця з обрамленням зберігається у class CSR::BD::Matrix в масивах вказівників на діагональні блоки та блоки з обрамленням. Тобто кожен блок є окремою підматрицею, збереженою у модифікованому форматі CSR (class CSR::Matrix).

Такий підхід дозволяє маніпулювати кожним блоком незалежно та забезпечувати можливість розпаралелення обчислень, як на MIMD-комп'ютерах, так і на комп'ютерах гібридної архітектури.

Class CSR::BD::Matrix зберігає інформацію про вхідну матрицю class MatrixInfo, а саме:

- розмірність матриці, кількість ненульових елементів та її розрідженість;
- кількість заданих перерізів при перетворенні матриці;
- кількість отриманих блоків у діагоналі та в обрамленні;
- характеристики матриці такі як симетричність, представлення матриці у трикутному вигляді, та місце розташування трикутника (над чи під головною діагоналлю);
- базис матриці (нульовий чи одиничний).

В свою чергу class CSR::Matrix зберігає підматриці у модифікованому форматі CSR, тобто поєднує у собі 3-х та 4-х векторний формат збереження, а саме зберігаються такі масиви:

- vector<double> Value – ненульові елементи матриці;
- vector<int> Columns – позиція ненульового елемента у рядку;

- vector<int> RowIndex – кількість ненульових елементів у кожному рядку;
- vector<int> PointerB – кількість ненульових елементів у кожному попередньому рядку;
- vector<int> PointerE – кількість ненульових елементів у кожному наступньому рядку;
- MatrixInfo csr_bd_matrix_info – містить інформацію про кожну окрему підматрицю та її розташування у блочно-діагональній матриці з обрамленням.

В результаті отримано уніфікований 5-ти рядковий формат збереження розрідженої матриці, що дозволяє використовувати відомі високопродуктивні бібліотеки обчислювальної математики. Крім того, якщо в процесі обчислень додаткові масиви не потрібні, то вони видаляться автоматично, тим самим досягається більш оптимальне використання оперативної пам'яті.

Такий формат є ефективним як для використання на CPU, так і на GPU. Він дає змогу, наприклад, застосовувати до блоків матриць в одній і тій же програмі, без проведення переіндексації їх елементів, як функції розпаралелення системи MPI, так і програми матрично-векторних операцій з бібліотеки MKL. Тим самим забезпечується більш ефективне використання обчислювальних ресурсів та висока масштабованість гібридних алгоритмів. Серед переваг обраного формату можна також зазначити прозорий підхід до проведення обчислень, передачі повідомлень між обчислювальними процесами, зменшення витрат часу на розробку, відлагодження та тестування гібридних програм.

Налаштування гібридних алгоритмів та програм на обчислювальні ресурси гібридного комп'ютера. В сучасних гібридних комп'ютерах відмічається постійне оновлення типів обчислювальних вузлів та відеокарт, розробляються нові версії систем паралельного програмування MPI та CUDA, а також бібліотек прикладних програм MKL, CuBLAS, CuSparse тощо. Використання найновіших розробок програмно-технічного устаткування сприяє підвищенню ефективності гібридних алгоритмів та програм. Тому необхідно передбачити такий спосіб їх розробки, щоб можна було легко вносити відповідні зміни зі змінами в обчислювальних ресурсах гібридного комп'ютера.

З цією метою використовується концепція «функцій-обгорток» [7]. «Обгортка» (англ. Wrapper) – це функція, яка є проміжною ланкою між прикладними функціями або програмами та іншою бібліотекою чи програмним інтерфейсом, а також може модифікувати або узагальнювати інтерфейси прикладних програм. При цьому реалізується модульний принцип програмування. Таким чином, при необхідності оновлення програмного забезпечення достатньо вносити поправки лише в окремі програмні модулі. Комбінований формат збереження блочно-діагональної розрідженої матриці з обрамленням дозволяє ефективно використовувати концепцію «функцій-обгорток».

До «обгорток» винесені службові функції для роботи з графічним прискорювачем, наприклад, `cusparseCreate(…)`, `cudaMemcpy(…)` та інші.

Блок-схему функціонального складу гібридної показано на рис. 2, до якого входять:

- `Data structures` – модуль, що відповідає за структури збереження даних та їх обробку;
- `Wrappers function` – модуль, в якому реалізовуються «функції-обгортки»;
- `Hight performance computing libraries` – зовнішні високопродуктивні бібліотеки та інтерфейси паралельного програмування;
- `Statistics functions` – модуль, що відповідає за збір, збереження та обробку статистичних даних, підвищення ефективності обчислювального процесу, а також за обробку виключних та помилкових ситуацій;
- `Computing methods functions` – модуль, в якому реалізовано обчислювальні алгоритми (для своєї роботи цей модуль використовує вище приведені модулі);
- `External user interface` – модуль, що реалізує інтерфейс користувача.

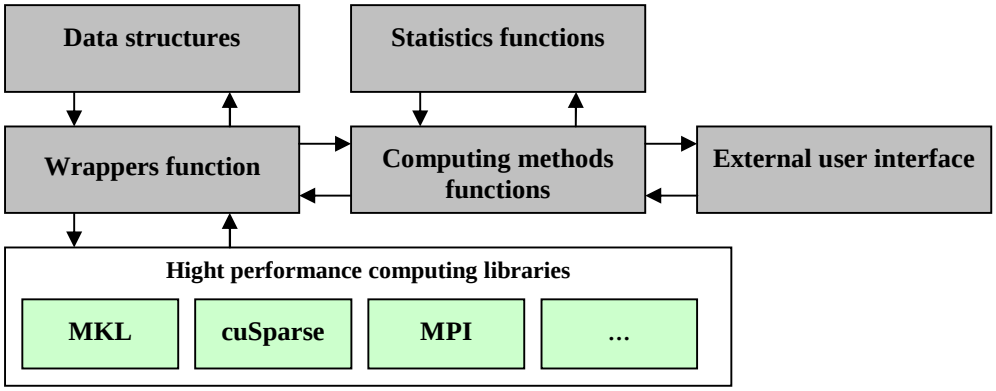

РИС. 2. Блок-схема гібридного програмного засобу

Такий підхід до розробки паралельного програмного забезпечення дозволяє отримати наступні переваги:

- розширюваність – за рахунок малої залежності між модулями можна легко додавати нові функціональні можливості, наприклад, підтримку нових високопродуктивних бібліотек або структур збереження даних;
- зменшення часу на відлагодження та супровід програмного засобу – відлагодження модулів відбувається лише один раз, і в разі повторного використання повторне тестування не потрібно проводити.

Експериментальне дослідження розроблених алгоритмів. Для дослідження алгоритмів використано розріджені матриці з колекції Флоридського університету [8], які приведено в табл. 1.

ТАБЛИЦЯ 1. Набір тестових розріджених матриць з Флоридської колекції

Задача	Порядок матриці	Кількість ненульових елементів
Bmwcra_1	148 770	10 641 602
Bone010	986 703	47 851 783
Emila_923	923 136	40 373 538

Обчислювальні експерименти проводилися на гібридному комп'ютері Інпарком_G [9] з такими технічними характеристиками: чотири вузли з двома 4-х-ядерними Intel Xeon E5606 процесорами, оперативна пам'ять: 3 Гб на одне фізичне обчислювальне ядро, 2 графічних процесора Nvidia Tesla M2090 на вузол.

В табл. 2 приведено порівняльна часова характеристика знаходження найменшого власного значення на гібридному комп'ютері паралельним попереміно-трикутним алгоритмом.

ТАБЛИЦЯ 2. Часова характеристика (сек) розв'язування задач на Інпарком_G

Задача	CPU				CPU + GPU			
	1 Core	8 Core	16 Core	32 Core	1GPU	2GPU	4 GPU	8 GPU
Bone010	423,8	55,76	31,68	18	67,06	29,47	17,86	11,91
Bmwcra_1	820,74	115,27	62,65	34,05	117,42	56,02	30,78	18,77
Emilia_923	1185,52	173,07	90,14	46,95	161,08	77,99	44,32	26,07

З табл. 2 видно, що за гібридним попереміно-трикутним алгоритмом (без використання GPU) найбільше прискорення у порівнянні з послідовною версією програми отримано від 23 до 25 разів для всіх розглянутих матриць. Використання одного графічного прискорювача дає прискорення обчислень в 6 – 7 разів у порівнянні з 1 CPU, а при масштабуванні системи до 8 графічних прискорювачів – в 35 – 45 разів у порівнянні з послідовною версією програми.

Висновки. Обчислення власних значень та відповідних векторів для розріджених матриць великих розмірностей потребують значних обчислювальних ресурсів. Показано, що структурна регуляризація вихідних розріджених матриць, ефективні алгоритми збереження та їх обробки, а також врахування особливостей архітектури гібридного комп'ютера та систем паралельного програмування забезпечують суттєве зменшення часу розв'язування задач та хороше масштабування.

А.В. Чистяков

ОБ ОСОБЕННОСТЯХ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ РЕШЕНИЯ ЗАДАЧ НА СОБСТВЕННЫЕ ЗНАЧЕНИЯ С РАЗРЕЖЕННЫМИ МАТРИЦАМИ НА ГИБРИДНЫХ КОМПЬЮТЕРАХ

Рассматриваются проблемы разработки параллельного алгоритмически-программного обеспечения для решения задач на собственные значения на компьютерах гибридной архитектуры. Предложены некоторые пути повышения эффективности параллельных алгоритмов на примере реализации гибридных итерационных алгоритмов для решения частичной алгебраической проблемы собственных значений для симметричных положительно-определенных матриц разреженной структуры.

A.V. Chystyakov

ABOUT THE FEATURES OF SOFTWARE DEVELOPMENT FOR SOLVING EIGENVALUE PROBLEMS FOR SPARSE MATRICES ON HYBRID COMPUTERS

The problems of the development of parallel algorithmic software for solving eigenvalue problems on hybrid computers is investigated. Some ways of improving the efficiency of parallel algorithms on the example of implementation of hybrid iterative algorithms for solving partial algebraic eigenvalue problem for symmetric positive-definite sparse matrices are presented.

1. *Хіміч О.М., Чистяков О.В.* Паралельні однокрокові ітераційні методи розв'язання алгебраїчної проблеми власних значень для розріджених матриць // Компьютерная математика. – 2014. – № 2. – С. 81 – 88.
2. *Message* Passing Interface (MPI). https://computing.llnl.gov/tutorials/mpi/.
3. *Боресков А.В., Харламов А.А.* Основы работы с технологией CUDA. – М.: Пресс, 2010. – С. 232.
4. *Math* Kernel Library – Intel. http://software.intel.com/en-us/intel-mkl.
5. CUBLAS Linear Algebra.http://developer.download.nvidia.com/CUBLAS Library.pdf
6. *Писсанецки С.* Технология разреженных матриц. – М.: Мир, 1988. – С. 410.
7. *Wrapper* function http://en.wikipedia.org/wiki/Wrapper_function
8. *The University* of Florida Sparse Matrix Collection. http://cise.ufl.edu/research/sparse/matrices.
9. *Сергиенко И.В., Молчанов И.Н., Перевозчикова О.Л., Химич А.Н.* Гетерогенные кластеры семейства Инпарком // Міжнародна конференція «Високопродуктивні обчислення», HPC-UA 2013, (Україна, Київ, 12 − 14 жовтня 2013).

Одержано 05.02.2015

Про автора:

Чистяков Олексій Валерійович,
аспірант Інституту кібернетики імені В.М. Глушкова НАН України.