УДК 681.03

ПРИНЦИПИ СТВОРЕННЯ ІНТЕЛЕКТУАЛЬНОГО ІНТЕРФЕЙСУ ДЛЯ РОЗВ'ЯЗУВАННЯ СИСТЕМ ЛІНІЙНИХ АЛГЕБРАЇЧНИХ РІВНЯНЬ НА КОМП'ЮТЕРАХ ГІБРИДНОЇ АРХІТЕКТУРИ

О.М. Хіміч, Т.В. Чистякова, А.Ю. Баранов

Інститут кібернетики ім. В.М. Глушкова НАН України,
Київ, проспект Академіка Глушкова 40, dept150@insyg.kiev.ua

Розглянуто принципи та методологію створення інтелектуального програмного засобу для автоматичного дослідження та розв'язування систем лінійних алгебраїчних рівнянь на паралельному комп'ютері гібридної архітектури, яка поєднує обчислення на багатопроцесорних комп'ютерах MIMD-архітектури з прискоренням обчислень на графічних процесорах SIMD-архітектури.

Methodology of intelligent software development for automatic researching and solving systems of linear algebraic equations on parallel hybrid computer, which combine computations on multicore computers with MIMD-architecture with accelerated computations on GPU with SIMD-architecture, is considered in this paper.

Проблеми розв'язування систем лінійних алгебраїчних рівнянь на паралельних комп'ютерах

У більшості задач інженерії та науки, які розв'язуються на сучасних комп'ютерах, проміжним або кінцевим етапом є розв'язування задач лінійної алгебри, зокрема, систем лінійних алгебраїчних рівнянь (СЛАР) з наближено заданими вихідними даними. Необхідно підкреслити, що якщо навіть спеціалісти-прикладники вважають, що їхня задача має точно задані вихідні дані, комп'ютерна модель задачі, яку в кінцевому рахунку і доводиться реалізовувати на комп'ютері, завжди має наближений за відношенням до математичної задачі характер, у тому числі і через похибки введення числових даних про задачу в комп'ютер [1].

Отже, комп'ютерні моделі задач слід розглядати як задачі з апріорі невідомими властивостями та проводити комп'ютерне дослідження математичних властивостей комп'ютерних моделей задач, а саме [1]:

- встановити існування і єдиність розв'язку комп'ютерної задачі;
- дослідити стійкість розв'язку в рамках рівня похибки переведення чисел з десяткової системи числення в комп'ютерну;
- визначити характерні властивості комп'ютерної моделі задачі для вибору ефективного алгоритму її розв'язування;
- оцінити спадкову похибку математичного розв'язку;
- оцінити обчислювальну похибку отриманого розв'язку, тобто близькості комп'ютерного розв'язку до точного розв'язку комп'ютерної задачі.

Для розв'язування задачі на комп'ютерах гібридної архітектури з багатоядерними CPU і багатьма GPU користувачеві необхідно проводити такі додаткові роботи:

- визначення необхідної кількості ядер CPU, процесорів GPU, та побудова ефективної топології міжпроцесорних зв'язків;
- забезпечення рівномірного завантаження процесорів, які використовуються для розв'язання задачі;
- синхронізація обмінів даними між процесорами;
- мінімізація комунікаційних втрат, зумовлених необхідністю обміну інформацією між CPU та GPU.

Така робота вимагає від користувачів навичок паралельного програмування з використанням технологій MPI [2] та CUDA [3], знань математичних і технічних особливостей комп'ютера зі складною гібридною архітектурою, вивчення великої кількості експлуатаційної документації пакетів і бібліотек, що реалізують паралельні алгоритми розв'язування задач з даної проблематики.

Вирішення вказаних проблем розв'язування СЛАР на комп'ютерах гібридної архітектури можна перекласти на інтелектуальний інтерфейс, що пропонується.

Принципи створення інтелектуального інтерфейсу

Основні принципи технологічної схеми розв'язування задач за допомогою інтелектуального інтерфейсу:

- можливість розв'язування задач з наближеними вихідними даними;
- постановка задачі на мові предметної області;
- природні для користувача форми введення вихідних даних задачі (файловий, програмний, клавіатурний);
- автоматизація процесів комп'ютерного дослідження математичних властивостей задачі, вибору алгоритму і синтезу програми розв'язування на основі знань про предметну область та про задачу, що розв'язується;

- розв'язування задачі з оцінкою достовірності отримуваних комп'ютерних результатів;
- отримання не тільки розв'язку задачі, але і протоколу процесу дослідження її властивостей та оцінок достовірності отриманих результатів;
- реалізація принципів «прихованого паралелізму»;
- отримання необхідної допомоги на кожному етапі роботи.
- Принципи «прихованого паралелізму» передбачають [4]:
- автоматичне створення ефективних паралельних алгоритмів розв'язування задач;
- автоматичне визначення необхідних обчислювальних пристроїв (ядер CPU та процесорів GPU), побудову ефективної конфігурації комп'ютера для даної задачі;
- автоматичне планування обчислювального процесу з ефективним використанням CPU та GPU;
- автоматичний розподіл вихідної інформації за процесами у відповідності до вимог кожного з алгоритмів;
- рівномірне завантаження процесів, що використовуються при розв'язуванні задачі;
- синхронізацію обмінів між процесами;
- мінімізацію обмінів між процесами.

Таким чином, для користувача забезпечується така технологія роботи з інтелектуальним інтерфейсом на комп'ютері гібридної архітектури з багатоядерними і графічними процесорами, як і на однопроцесорному комп'ютері.

Інтелектуальний інтерфейс, реалізується за формальною моделлю предметної області [5]. Його розробка заснована на синтезі основних досягнень в області модульного програмування, баз даних, баз знань і спирається на розвинуті методи обробки знань: їх подання, зберігання, отримання нових знань і т. д.

Модель предметної області

Під предметною областю, в даному випадку, будемо розуміти сукупність задач, які пов'язані з дослідженням та розв'язуванням СЛАР та пов'язаних з ними задач, алгоритми їх комп'ютерного дослідження, розв'язування та аналізу достовірності отримуваних результатів.

Модель предметної області – формалізований набір знань про об'єкти предметної області та відношення між ними.

Конструювання інтелектуального інтерфейсу на моделі предметної області включає:

- визначення задач, які передбачається реалізовувати, та алгоритмів їх дослідження та розв'язування;
- розробку комп'ютерної технології планування обчислювального процесу: автоматичного дослідження властивостей задач та побудову відповідного алгоритму її розв'язування з аналізом достовірності результатів, визначення необхідних ядер CPU та процесорів GPU, синтез програми розв'язування і т. д.;
- побудову формальної моделі предметної області, використовуючи різні способи подання та застосування знань про задачі, алгоритми розв'язування, а також з урахуванням необхідних обчислювальних засобів;
- побудову моделі та форм спілкування з користувачем.

Принципи комп'ютерного дослідження СЛАР та пов'язаних з ними задач з метою автоматичної побудови відповідного алгоритму та синтезу програми розв'язування на необхідних обчислювачах потребують таких підходів до створення інтелектуального інтерфейсу, щоб комп'ютер застосовувався вже на етапі постановки задачі на мові предметної області.

Дослідження задачі з даної предметної області включає:

- аналіз параметрів задачі, достатніх для організації автоматичного дослідження її властивостей і розв'язування;
- аналіз складу алгоритмів для проведення дослідження та розв'язування задачі;
- встановлення логічних зв'язків та можливих переходів при дослідженні задачі та побудові алгоритму і програми розв'язування;
- аналіз необхідних ядер CPU та процесорів GPU для організації обчислювального процесу, взаємозв'язків з операційним середовищем та користувачем.

Предметна область з даного класу включає в себе широкий спектр задач з урахуванням наближеного характеру вихідних даних:

- розв'язування СЛАР та визначення спадкової і обчислювальної похибок комп'ютерного розв'язку;
- обернення матриці;
- псевдообернення матриці;
- обчислення оцінки числа обумовленості матриці;
- обчислення рангу матриці та інші.

В результаті проведеного дослідження ефективності різних алгоритмів розв'язування СЛАР було вибрано для реалізації блочні алгоритми: *LU*-розвинення матриці, LL^T-розвинення матриці та сингулярного (SVD) розвинення, які можуть забезпечити найкращу збалансовану обробку матриць на комп'ютерах гібридної архітектури [1]. Реалізація алгоритмів зводиться до виконання алгебраїчних дій над окремими блоками матриць, на які вони розподіляються, визначивши розміри блоків у відповідності з розмірами кеш-пам'яті

процесорів CPU та розмірами розподіленої пам'яті GPU. Ще однією перевагою зазначених алгоритмів є можливість використовувати вже відомі бібліотеки стандартних програм, наприклад BLAS [6], LAPACK [7], CUBLAS [8], що реалізують з великою швидкодією матрично-векторні операції на CPU та GPU. Це дає змогу значно підвищити швидкодію алгоритмів.

Модульний аналіз предметної області і модульний принцип програмування [9] дають можливість розбити задачі на окремі під-задачі, та розробити відповідні функціональні програмні модулі, що їх реалізують на різних обчислювальних пристроях, а також провести систематизацію та уніфікацію знань з предметної області і розробити однотипні і спеціальні способи зберігання, пошуку, вилучення та обробки знань.

Кожен функціональний модуль містить знання про можливість та правила його використання, про вхідні та вихідні параметри, правила розподілу вихідних даних за процесами, про допустиму топологію комп'ютера, необхідні ядра CPU та процесори GPU і т. д. Знання про властивості задачі та про алгоритми, програми, ядра CPU та GPU, які необхідно використати для ефективного її розв'язування, отримуються в процесі дослідження.

Формальна модель предметної області описується семантикою функціональних модулів, що має вигляд орієнтованого функціонального графа, вершини якого відповідають поняттям і об'єктам предметної області, а дуги – відносинам між ними (рис. 1).

Вершини цього графа є таких видів:

- вершини-присвоєння, тобто вхідні дані задачі, що задаються користувачем, та властивості задачі, які виявляються в процесі дослідження та формалізуються через спеціальні коди (S_1, S_4, S_5 $S_{12,}$);
- вершини-функціональні модулі, що реалізують логічно закінченні частини алгоритмів (S_{11}, S_6, S_7, S_8, S_9, S_4, S_5, $S_{11,}$ S_{15} S_{16}, $S_{19,}$ S_{20});
- вершини-перемикачі, що моделюють перехід на інші задачі, та режими розв'язування задачі (P_1, P_2, P_3, S_1);
- вершини-діалог з користувачем ($S_{11,}$ S_{15} S_{16}, $S_{19,}$ S_{20});
- вершини-результат, де користувач одержує детальне пояснення ходу обчислювального процесу та візуалізує розв'язок з оцінками його достовірності (S_{10} S_{13}, $S_{21,}$ S_{22}).

Дуги графа є двох видів:

1) інформаційні, що визначають порядок проходження за графом при повній специфікації (пунктирними лініями позначаються дуги, які навантажуються інформацією від користувача);

2) логічні, за якими порядок проходження за графом визначається в результаті дослідження в вершинах-функціональних модулях.

Крім того, на рис. 1 ми маємо такі позначення:

- вхідні дані задачі:
  - N – кількість рядків матриці*;*
  - M – кількість стовпчиків матриці*;*
  - A – масив, що містить елементи матриці;
  - B – масив, що містить елементи правої частини СЛАР;
  - EA, EB – максимальні відносні похибки елементів матриці та правої частини відповідно;
- властивості матриці СЛАР, які виявлено в процесі дослідження:
  - Not positiv definit – матриця СЛАР не є додатно означеною;
  - Syng – матриця СЛАР є машинно виродженою;
- результати обчислень:
  - X – вектор розв'язку СЛАР;
  - *cond* – оцінка числа обумовленості матриці;
  - *rang* – ранг матриці;
  - E_1, E_2 – оцінка спадкової та обчислювальної похибки відповідно.

Передбачається автоматичне розв'язування задачі – без втручання користувача в обчислювальний процес, та інтерактивне – з деякою участю користувача: надається можливість вибрати кількість процесів, призупинити обчислення і т. д.

Графовий спосіб формалізації знань про предметну область дає можливість детально описати різні технологічні схеми обчислювального процесу та взаємодії з користувачем, розглянути велику кількість варіантів ухвалення рішень. Водночас надається можливість швидко знайти на графі всієї формальної моделі за певними властивостями задачі найбільш короткий шлях до пошуку розв'язку. Крім того, графовий спосіб формалізації предметної області є найбільш придатним для створення спеціальних та використання готових інструментальних засобів для програмування.

На основі семантики функціональних модулів і у відповідності до знань про зв'язки між ними здійснюється планування обчислювального процесу. Застосовуються статичний і динамічний способи планування обчислень. Принципова їхня відмінність полягає у ступені визначеності прийняття рішень при реалізації поставленої задачі. Якщо такої можливості немає, тобто коли черговий крок обчислень можна спланувати тільки після завершення поточного кроку, має застосовуватися динамічне планування (наприклад,

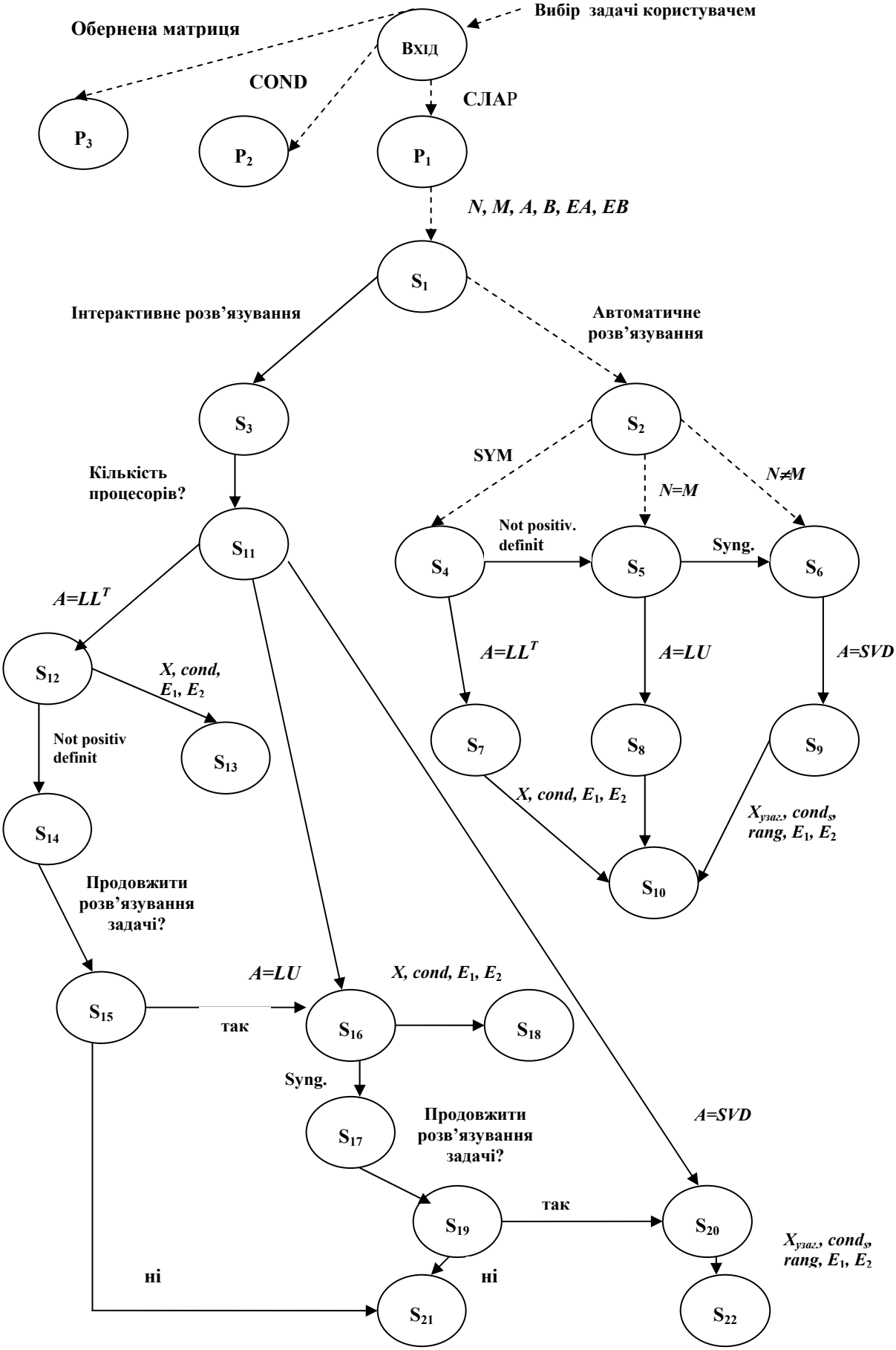

Рис. 1. Фрагмент формальної моделі розв'язування СЛАР в інтелектуальному інтерфейсі

в результаті дослідження властивостей комп'ютерної моделі задачі динамічно планується автоматична побудова алгоритму, що відповідає цим властивостям, і синтез програми розв'язування).

Деякі особливості створення паралельних алгоритмів та програм для розв'язування СЛАР на комп'ютерах гібридної архітектури

Основні проблеми паралельних обчислень на комп'ютерах гібридної архітектури пов'язані з узгодженням розподілу під-задач та даних на CPU та GPU, а також оптимізацією комунікаційних витрат при цьому. Ефективність за часом розв'язування однієї і тієї ж задачі на паралельних комп'ютерах різних архітектур може бути різною. Це залежить від того, наскільки можна врахувати математичні та технічні особливості комп'ютера при створенні алгоритмів та програм, що реалізують цю задачу. Таким чином, виникає проблема дослідження найбільш ефективної архітектури комп'ютера для конкретної задачі: MIMD, SIMD, гібридної; пропускної спроможності мережі, співвідношення обчислювальних характеристик GPU та CPU; різних способів організації пам'яті: загальна, розподілена, багаторівнева, з багатьма конфігураціями зв'язків між процесорами: кільце, решітка, гіперкуб, дерево.

Необхідно також враховувати особливості програмних засобів, які використовуються для розпаралелювання: MPI – для розпаралелювання між процесами на ядрах CPU, CUDA – для розпаралелювання на GPU.

При створені паралельних алгоритмів дослідження властивостей СЛАР та їх розв'язування з оцінками достовірності комп'ютерних результатів на комп'ютерах гібридної архітектури необхідно вирішити такі проблеми:

- Визначити для яких видів матриць ефективно використовувати гібридну архітектуру;
- розподілити задачі на під-задачі з метою ефективного розпаралелення на CPU та GPU гібридної системи;
- визначити необхідні обчислювальні ресурси;
- врахувати наявність кеш-пам'яті процесорів для забезпечення високої швидкодії алгоритму;
- забезпечити рівномірне завантаження (балансування) обчислювальних;
- оцінити ефективність алгоритму.

Розподіл задачі на під-задачі здійснюється з метою визначення: пріоритетів під-задач, на яких обчислювальних пристроях їх краще реалізовувати за часом виконання, які під-задачі можна виконати паралельно, в якій послідовності їх треба виконувати і т. д.

Обсяг обчислень для кожного обчислювального пристрою, що використовується, має бути приблизно однаковий – це дозволить забезпечити їх рівномірне обчислювальне завантаження (балансування). Крім того, також зрозуміло, що розподіл під-задач між обчислювальними пристроями має бути виконано таким чином, щоб кількість інформаційних зв'язків (комунікаційних взаємодій) між під-задачами була мінімальною.

При розробці алгоритмів для гібридної архітектури необхідно враховувати відмінність в організації пам'яті на CPU та GPU. Наприклад, швидкодію алгоритмів та програм на CPU можна значно підвищити, якщо розробляти їх з урахуванням наявної кеш-пам'яті процесора. А на GPU є 6 видів пам'яті, кожна з яких має своє призначення. Розподілена пам'ять (shared memory) – це швидка пам'ять, яку доцільно використовувати як кеш при розв'язуванні задач. Проте ця пам'ять дуже мала у порівнянні з кеш-пам'яттю CPU. На один мультипроцесор доступно всього 16 Кбайт розподіленої пам'яті. Отже, величина блоків матриць для їх обробки на CPU та GPU буде різною, з урахуванням обсягу "швидкої пам'яті".

Планування обчислень в інтелектуальному інтерфейсі

В інтелектуальному інтерфейсі передбачається дворівневе планування обчислень:

1-й рівень – автоматичне дослідження математичних властивостей комп'ютерної задачі і вибір відповідного алгоритму розв'язування з оцінками достовірності (оцінка спадкової похибки в математичному розв'язку і оцінка обчислювальної похибки) [1];

2-й рівень – автоматичне планування реалізації конкретного алгоритму між ядрами CPU та процесорами GPU гібридної системи з метою найменших витрат комп'ютерного часу і потужностей гібридних компонентів.

Перший рівень планування обчислень реалізується на основі розробленого математичного апарату автоматичного дослідження і розв'язування СЛАР і пов'язаних з ними задач з наближено заданими вихідними даними, який представляється у вигляді графа формальної моделі. Програмна реалізація цієї частини інтелектуального інтерфейсу виконана за допомогою об'єктно-орієнтованого програмування.

Коротко розглянемо як реалізується автоматичне планування обчислень першого рівня (рис. 1):

1) Якщо введена в комп'ютер матриця системи симетрична, то для її дослідження автоматично вибирається алгоритм LL^T-розвинення. При цьому визначаються необхідні топологія комп'ютера та кількість ядер CPU;

2) Якщо матриця системи в комп'ютері виявилася додатно означеною, то знаходиться розв'язок задачі з оцінками достовірності, в іншому випадку – матриця досліджується на виродженість алгоритмом *LU*-розвинення;

3) Якщо матриця системи в комп'ютері виявилася невиродженою, то знаходиться розв'язок задачі з оцінками достовірності, інакше – розв'язування задачі продовжується алгоритмом SVD-розвинення. В результаті розв'язування користувач отримує протокол обчислювального процесу, розв'язок задачі з оцінками достовірності та деякі інші характеристики задачі.

Автоматичне планування другого рівня передбачає аналіз задачі з точки зору її декомпозиції на під-задачі з метою розподілу під-задач, функціональних модулів, що їх реалізують, та даних на необхідні обчислювальні пристрої для реалізації алгоритму розв'язування: тільки на CPU, на одному процесорному ядрі CPU з використанням прискорювача GPU, на декількох процесорах CPU з використанням прискорювачів GPU.

Розглянемо як виконується автоматичне планування другого рівня на прикладі блочного алгоритму LL^T-розвинення.

Коротко опишемо алгоритм. При розв'язуванні СЛАР $Ax = b$ з симетричною додатно означеною матрицею алгоритм LL^T-розвинення приводить матрицю A до виду $A = LL^T$, де L – нижня трикутна, а L^T – верхня трикутна матриці.

Для реалізації паралельного блочного алгоритму будемо вважати, що матриці A і L порядку n розділено на квадратні блоки порядку s.

На k-му кроці розглянутого блочного алгоритму LL^T-розвинення ($k = 1, 2, \ldots$) підматрицю (діагональний блок матриці A) $A^{(k)}$ порядку $r = n-(k-1)s$, яка містить останні r рядків і r стовпчиків матриці A, можна представити в такому вигляді:

$$\begin{pmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} L^T{}_{11} & L^T{}_{21} \\ 0 & L_{22} \end{pmatrix} = \begin{pmatrix} L_{11}L^T{}_{11} & L_{11}L^T{}_{21} \\ L_{21}L^T{}_{11} & L_{21}L^T{}_{21} + L_{22}L^T{}_{22} \end{pmatrix},$$

де блок A_{11} має розмірність $s \times s$, блок A_{12} – розмірність $s \times (r-s)$, блок A_{21} – розмірність $(r-s) \times s$, блок A_{22} – розмірність $(r-s) \times (r-s)$.

Для реалізації алгоритму на кожному кроці необхідно:

- виконати LL^T-розвинення матриці A_{11} і отримати таким чином матрицю L_{11};
- обчислити матрицю L_{21} за формулою:

$$L_{21} = A_{21} \cdot \left(L_{11}{}^T\right)^{-1};$$

- модифікувати матрицю A_{22} за формулою:

$$\tilde{A}_{22} = A_{22} - L_{21} \cdot L_{21}{}^T = L_{22} \cdot L_{22}^T.$$

Отже, на k-му кроці отримується модифікована частина матриці L. Далі, значення k збільшується на одиницю і повторюються обчислення для нового значення k, розглядаючи замість матриці $A^{(k)}$ матрицю $\tilde{A}_{22}$ (рис. 2).

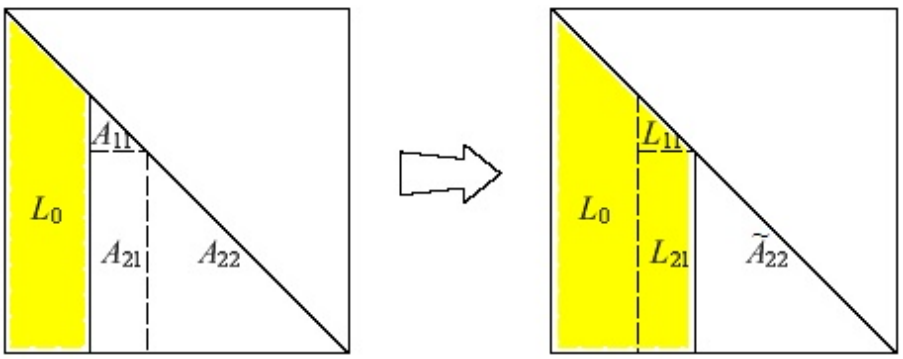

Рис. 2. Схема реалізації блочного алгоритму LL^T-розвинення

За основу цієї обчислювальної схеми на комп'ютері беремо схему реалізації блочного алгоритму LL^T-розвинення, яку реалізовано в пакеті програм LAPACK при використанні BLAS.

За цією схемою задача розподіляється на такі підзадачі:

LL^T– розвинення діагонального ведучого блоку матриці (модуль xPOTRF);

xSYRK – модифікація діагональних (нижніх трикутних) блоків матриці;

xGEMM – модифікація піддіагональних блоків матриці, які знаходяться справа від ведучого блоку;

xTRSM – модифікація блоків матриці, які знаходяться в одному стовпчику з ведучим блоком.

Псевдокод реалізації $A=LL^T$-розвинення матриці має вигляд:

```
for (k = 0; k  < Nt; k++)
A[k][k] <– xPOTRF(A[k][k])
for (m = k+1; m  < Nt; m++)
A[m][k] <– xTRSM(A[k][k],A[m][k])
for (n = k+1; n  < Nt; n++)
A[n][n] <– xSYRK(A[n][k],A[n][n])
for (m = n+1; m  < Nt; m++)
A[m][n] <– xGEMM(A[m][k],A[n][k], A[m][n])
```

Як ми бачимо з псевдокоду, операція в другому рядку може бути виконана раніше, ніж був повністю виконаний попередній крок в циклі. Для початку її виконання достатньо, щоб на попередньому кроці було визначено $A[k][k]$. Аналогічні міркування можна привести і для інших операцій. Таким чином, щоб приступити до виконання операцій на кроці k циклу, не обов'язково, щоб крок k-1 був завершений. Достатньо, щоб були отримані необхідні для кожної конкретної операції блоки матриці.

Таким чином, паралелізм задачі може бути здійснено використовуючи коди програм LAPACK та BLAS, із зазначенням частин алгоритмів (підзадач), які реалізуються на різних обчислювальних компонентах гібрида: тільки на ядрі CPU, тільки на GPU і на обох пристроях одночасно (гібридне завдання).

Планування реалізації цього алгоритму складається з виконання таких процедур:

- розподіл задачі на підзадачі;
- паралельні підзадачі (xTRSM, xSYRK, xGEMM) "плануються" для ефективного виконання на GPU з урахуванням асинхронності, величини блоку і т. д.;
- послідовна підзадача xPOTRF – розвинення діагонального блоку планується виконуватися на CPU (величина блоку з урахуванням кеш-пам'яті);
- малі завдання на CPU частково перекриваються великими завданнями на GPU (перемноження матриць).

Отже, алгоритм LL^T-розвинення можно представити у вигляді направленного ациклічного графа, який показано на рис. 3.

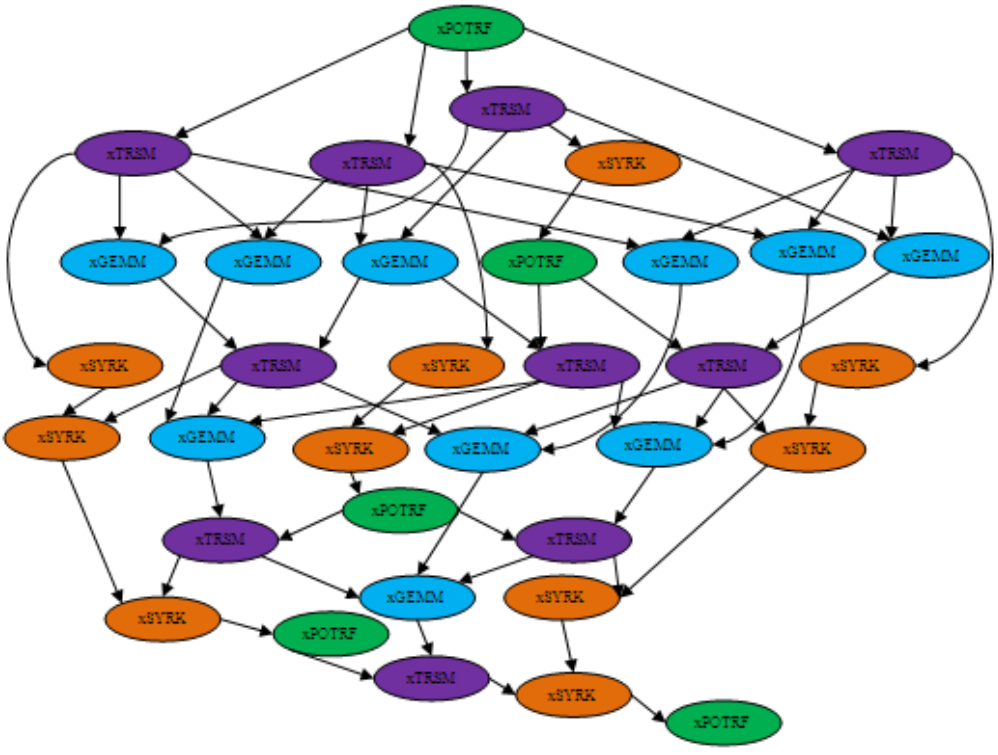

Рис. 3. Графічне представлення алгоритму LL^T-розвинення

Для планування обчислень другого рівня в інтелектуальному інтерфейсі застосовується інструментальний засіб StarPU [10, 11], який дає можливість заздалегідь оголосити всі операції, які будуть потрібні для виконання алгоритму, а також визначити між ними інформаційні залежності. Всі операції будуть виконуватися в тому

порядку і на тій кількості ядер CPU та процесорів GPU, які забезпечать найвищу ефективність реалізації алгоритму.

StarPU динамічно планує обчислення з урахуванням витрат часу виконання кожної підзадачі на кожному доступному обчислювальному пристрої. При цьому автоматично виконується асинхронне копіювання необхідних даних як між CPU і GPU, так і між GPU. Вихідні дані задачі один раз реєструються в середовищі StarPU, і надалі система сама, в разі необхідності, буде копіювати необхідні дані між CPU і GPU, максимально перекриваючи час виконання комутаційних зв'язків часом виконання математичних операцій.

Висновки

Багато науково-технічних задач зводяться до розв'язування систем лінійних алгебраїчних рівнянь великої розмірності. Для ефективного розв'язування таких задач необхідно використовувати паралельні комп'ютери, зокрема комп'ютери гібридної архітектури. В умовах наближених вихідних даних властивості комп'ютерних моделей апріорі не відомі. Тому необхідні нові підходи у створенні програмного забезпечення для гібридних систем, які б забезпечували достовірним комп'ютерним розв'язком при ефективному використанні обчислювальних ресурсів CPU та GPU.

Цю мету реалізує інтелектуальний інтерфейс з функцією автоматичної адаптації алгоритму, програми та архітектури гібридної системи на властивості задачі, що розв'язується, забезпечуючи при цьому як ефективне використання комп'ютерних ресурсів, так і оцінками достовірності комп'ютерних результатів.

1. *Химич А.Н., Молчанов И.Н., Попов А.В., и др.* Параллельные алгоритмы решения задач вычислительной математики. – Киев: Наук. думка, 2008. – 248 с.
2. *MPI*. http://www.netlib.org/mpi/
3. *Боресков А.В., Харламов А.А.* Основы работы с технологией CUDA.–М.: ДМК Пресс, 2010. – 232 с.
4. *Химич А.Н., Молчанов И.Н., Мова В.И. и др.* Численное программное обеспечение интеллектуального MIMD- компьютера. – Киев, Наук. думка. – 2007. – 216 с.
5. Представление знаний в человеко-машинных и робототехнических системах: В 3 т. – М.: ВИНИТИ, ВЦ АН СССР, 1984. – Т. А. – 216 с.; Т. В. – 236 с.; Т. С. – 378 с.
6. *BLAS* (Basic Linear Algebra Subprograms. http://www.netlib.org/blas/.
7. *LAPACK* – Linear Algebra PACKage. http://www.netlib.org/lapack/.
8. *CUBLAS Library*. http://developer.download.nvidia.com/compute/cuda/4_0/toolkit/docs /CUBLAS_Library.pdf.
9. *Поспелов Д.А.* Ситуационное управление. Теория и практика. – М.: Наука, 1986.
10. *Augonnet C., Thibault H., Namyst R., Wacrenier P.* StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. Concurrency and Computation: Practice and Experience, Euro-Par 2009 best papers issue, 2010. Accepted for publication.
11. *Topcuoglu H., Hariri S. and Min-You Wu.* Performance-effective and low-complexity task scheduling for heterogeneous computing. Parallel and Distributed Systems, IEEE Transactions on.– 2002 – 13(3) – P. 260–274.