УДК 681.5

АВТОМАТИЧНА ОПТИМІЗАЦІЯ ВИКОНАННЯ ДЛЯ ЗАДАЧІ МЕТЕОРОЛОГІЧНОГО ПРОГНОЗУВАННЯ

П.А. Іваненко, А.Ю. Дорошенко

Київський національний університет імені Тараса Шевченка, факультет кібернетики,
03187, Київ, проспект Академіка Глушкова 2, корпус 6, paiv@ukr.net

У статті розглядається поняття автотюнінгу – сучасного підходу до автоматичної оптимізації обчислювальних задач високої складності. На прикладі прикладної задачі метеорологічного прогнозування демонструється універсалізм та доцільність поєднання двох головних концепцій автотюнінгу, а саме: емпіричного й визначеного керування. Наводяться кількісні та якісні оцінки результатів швидкодії, отриманих у результаті чисельного експерименту.

This article describes idea of autotuning – a modern approach to solving problem of optimization of resource-intensive software. By example of meteorological forecast task the flexibility and efficiency of mixing various autotuning conceptions, such as empirical and defined control, are demonstrated. Quantitative and qualitative results of experiment are examined and possible directions for further research are considered.

Вступ

Сучасні прикладні наукові задачі вимагають значних обчислювальних ресурсів, тому задача оптимізації прикладних задач у багатопроцесорних середовищах займає чи не найперше місце в процесі розробки таких високопродуктивних програм. Ефективне виконання обчислень вимагає використання ефективних паралельних алгоритмів, швидкодія яких у свою чергу залежить від конкретного середовища виконання. Необхідність оптимізації виникає тому, що на етапі проектування застосунка частіше за все інформація про середовище виконання відсутня. І навіть якщо конфігурація відома, то майже неможливо визначити таку конфігурацію програми, за якої обчислення будуть виконуватися з максимально ефективним використанням усіх процесорів, пам'яті та кешів. Також слід пам'ятати, що програма, оптимізована під конкретну обчислювальну платформу, не зможе показати максимальну швидкодію на будь-якій іншій відмінній платформі. Саме тому зазвичай усі релевантні для швидкодії параметри виносяться в деяку зовнішню конфігурацію, значення якої змінюють під час оптимізації. Складність такої оптимізації полягає в тому, що зв'язок між цими параметрами не є тривіальним, тому оптимізація майже завжди зводиться до повного перебору усіх можливих конфігурацій. За виключенням тривіальних задач кількість та область значень цих параметрів визначають множину усіх можливих конфігурацій значної потужності, що робить такий підхід неможливим без автоматизації. Також слід зауважити, що часто оптимальна конфігурація не є інтуїтивною.

Одним із традиційних підходів до вирішення задачі оптимізації вважається використання «паралельних» компіляторів. Незважаючи на високу інтелектуальність таких компіляторів й непогані результати швидкодії, цей підхід утратив популярність через обмеження, які він накладав на проектування програм, а також через ускладнення й урізноманітнення архітектури мікропроцесорів, яке має місце в останні декілька десятиріч.

Відносно нова парадигма програмної автоматичної оптимізації (для зручності будемо використовувати скорочений термін «автотюнінг») уважається перспективним програмним підходом, який дозволяє науковцям та інженерам ефективно використовувати безперервно зростаючі обчислювальні потужності суперкомп'ютерів з точки зору витраченого часу й енергії. Сфера її застосування простягається від наукових до загально-цільових обчислень й не обмежується суперкомп'ютерами. Під клас застосунків, оптимізація яких може виконуватися авто тюнерами, підпадають ресурсоємні прикладні програми, швидкість й ефективність виконання яких залежить від конфігурації обчислювального середовища (кількість процесорів, об'єм оперативної пам'яті, розмір кешів процесора та пристроїв зчитування інформації, пропускна здатність мережі, тощо) й які мають механізми адаптації чи дозволяють легко їх інтегрувати. Фактично цей клас задач включає усі нетривіальні застосунки для будь-яких середовищ – від платформ для розподілених обчислень до мобільних пристроїв. Автотюнери успішно розв'язують описану вище проблему, а саме: дозволяють створювати максимально ефективні в будь-яких обчислювальних середовищах застосунки з мінімальним людським втручанням у процес оптимізації.

У цій роботі будуть широко розглянуті поняття й загальні методи автотюнінгу, а також буде надано результат їх застосування до розв'язання задачі автоматичної оптимізації прикладного паралельного застосунка. Для практичного розгляду була обрана задача короткочасного метеорологічного прогнозування, оскільки вона має високу обчислювальну складність і природну необхідність у максимально швидкому й точному результаті. Чисельний метод розв'язання цієї задачі дозволяє також наочно продемонструвати ефективність одночасного використання концептуально різних підходів до автотюнінгу, а саме: поєднання методологій визначеного та емпіричного керування (далі вони будуть розглянуті детальніше).

Основні концепції автотюнінгу

Розглянемо спочатку саме поняття автоматичної програмної оптимізації. Слово «програмний» свідчить про те, що ця парадигма реалізується в програмних технологіях. Тобто суб'єктом оптимізації є програмне забезпечення (далі ПЗ). Хоча в переважній більшості випадків об'єктом тюнінгу є також ПЗ, у загальному ним можуть бути й деякі апаратні засоби. При подальшому розгляді будемо вважати, що в ролі як суб'єкта, так і об'єкта буде виступати ПЗ, тобто програмне забезпечення оптимізує програмне забезпечення.

Слово «автоматичний» підкреслює необхідність максимальної автоматизації процесу, проте не виключає повністю участі розробників. Програмна оптимізація повинна звільняти досвідчених розробників від необхідності виконувати довготривалу й не високоінтелектуальну роботу по оптимізації ПЗ. Однак взаємодія з розробником необхідна, оскільки тільки людина може дати кінцеву оцінку швидкодії та ефективності. Тому вважається, що, крім швидкодії, є другий об'єкт оптимізації – швидкість розробки ПЗ, яка потребує компромісу.

Останнє слово «оптимізація» вимагає виконання двох умов. По-перше, цільове ПЗ повинно бути адаптованим. По-друге, має існувати автоматичний механізм, який буде виконувати описану оптимізацію. Наведемо класифікацію автотюнерів, а також їх суттєві характеристики.

Як вже було зазначено, для програмної оптимізації програмне забезпечення повинно буди адаптованим й надавати можливість керування своєю адаптивністю. Існує декілька можливих підходів до адаптації застосунків: використання *новітніх мов програмування*, які автоматично надають інтерфейс для адаптації ПЗ; *засоби автоматичної трансформації вихідного коду*; *розширення мов програмування* або *програмні середовища*, які допомагають розробникам у написанні адаптивного коду; й для повноти переліку наведемо також *ручну трансформацію коду*.

Розглянемо детальніше підхід, який інструментом налаштування ПЗ використовує перетворення вихідного коду. Зазвичай його поділяють на чотири категорії. Трансформації коду із першої категорії *(варіації планування)* не змінюють структури даних та розрахунки. Під цю категорію підпадають усі класичні перетворення циклів (розгортка, фрагментація тощо), а також конвеєрна обробка команд. Планування виконання задач й маскування затримок паралельних обчислень (вибірка з випередженням) також підпадають під цю категорію.

Друга категорія охоплює техніки для маніпуляції *структурами даних*. Найпростішим прикладом є вибір між розгорткою двовимірних матриць за стовпцями чи рядками. Для паралельних обчислень з розподіленою пам'яттю прикладом може бути розподіл і декомпозиція даних між процесорами.

До третьої категорії включають *маніпулювання алгоритмами*. Наприклад, існує розмаїття алгоритмів для розв'язання розріджених лінійних рівнянь чи знаходження власних чисел матриць. Різні алгоритми можуть обчислювати результати з відносно невеликою похибкою, що може буди прийнятним. У такому випадку можна обрати той алгоритм, який виконує обчислення найшвидше.

До четвертої категорії зарахуємо трансформації, необхідні для користування особливостями апаратних засобів, наприклад SIMD архітектури, гетерогенні мультипроцесори, платформи для паралельного обчислення, графічні прискорювачі тощо.

Повернемося до розгляду можливих підходів до адаптації ПЗ. Автоматичне перетворення вихідного коду є найефективнішим для *локальних перетворень*, тобто таких перетворень, які зводяться до локальних модифікацій вихідного коду, наприклад трансформації циклів чи використання SIMD-інструкцій. Такі перетворення, як розпаралелення послідовного коду (наприклад з використанням MPI фреймворку), відносяться до *глобальних перетворень* й вимагають глибокого аналізу коду, що іноді нераціонально чи навіть неможливо. Використання нової мови програмування, створеної спеціально для автотюнінгу [1], дозволяє позбутися цього обмеження, проте вимагає додаткових затрат на переписування існуючого ПЗ. А такі перетворення, як маніпулювання алгоритмами чи структурами даних, важко реалізувати без наявного опису можливих варіацій у вихідному коді. Саме тому, незалежно від типу ПЗ, варто розглянути комбінування різних підходів до автотюнінгу.

Задача оптимізаційного механізму полягає в керуванні механізмами адаптації цільового ПЗ під конкретні умови обчислювального середовища. Є дві методології такого керування – *визначене керування* й *емпіричне керування*. У випадку *визначеного керування* спочатку визначають параметри, що впливають на швидкодію ПЗ. Також обираються деякі явні характеристики, які значною мірою моделюють очікувані умови виконання обчислень. Далі аналізується взаємозв'язок цих характеристик і параметрів, а також їх вплив на загальну продуктивність ПЗ. За результатами цього аналізу залежно від характеристик цільового середовища виконання обирається оптимальна конфігурація параметрів. У випадку *емпіричного керування* виконується аналіз продуктивності в цільовому середовищі, де залежно від різних значень параметрів обчислень обирається найбільш швидкодіюча конфігурація.

Перевагою емпіричного керування є відсутність потреби моделювання умов виконання, а також аналізу їх впливу разом з характеристиками адаптивності ПЗ на швидкодію обчислень. Таким чином, емпіричне керування робить можливим адаптацію ПЗ до наявних властивостей обчислювального середовища. Ця властивість є настільки важливою, що емпіричне керування вважається основною методологією автотюнінгу. Проте слід зауважити, що емпіричним керуванням не можна вирішити будь-яку задачу автотюнінгу. Слабкість цього підходу полягає в неможливості адаптації ПЗ до будь-яких динамічних умов. Наприклад, проактивний

автотюнінг можливий тільки за використання визначеного керування, тому в багатьох випадках необхідне поєднання обох механізмів керування.

Також слід згадати інший дуже важливий аспект автотюнінгу, а саме: його ефективність. Повний пошук серед усіх можливих комбінацій оптимізаційних технік, описаних вище, непрактичний, тому виникає необхідність скорочення дерева пошуку. Класичним підходом до скорочення області пошуку при «ручній» оптимізації є профілювання, яке визначає найбільш затратні фрагменти обчислень стосовно часу виконання. Така евристика схожа на визначення пріоритетів гілок у пошуковому дереві. Ідея представлення експертних знань розробників у вигляді пошукових евристик [2] широко застосовується у спеціалізованих, пристосованих для автотюнінгу, фреймворках [3] і мовах програмування [1]. Також широко застосованими підходами до оптимізації пошуку розглядаються пошукові алгоритми [4] й машинне навчання [5, 6].

Опис математичної моделі

Наведемо стислий опис моделі, декомпозиції області обчислень та методу розщеплення, щоб надати загальне уявлення про задачу. За детальнішим описом варто звернутися до [7].

Спочатку визначимося із вертикальною протяжністю розрахункової області. Від цього безпосередньо залежить остаточний вигляд рівнянь. Оскільки висота гір у переважній більшості випадків не перевищує 7000 м, то оберемо для моделювання висоти $z \in [8000;18000]$. Такий вибір вертикальної протяжності дозволить не враховувати вплив рельєфу на динаміку руху повітря, що відчутно спрощує модель. Отже, обмежимося розглядом частини вільної сухої атмосфери та не будемо здійснювати перехід до нової вертикальної координати, пов'язаної з рельєфом чи тиском.

Розглянемо модель циркуляції атмосфери, де невідомими величинами є: вектор швидкості V, абсолютна температура середовища T та густина повітря. Значення тиску p уважатимемо відомими, як і почактово-крайові умови впродовж усього часу розв'язання задачі. Як загально відомо, стан будь-якого газу визначається трьома параметрами: абсолютною температурою T, тиском p та густиною ρ

Рівнянням для визначення густини є рівняння стану повітря (рівняння Клапейрона) (1).

$$p = \rho R_C T, \tag{1}$$

де $R_C = 287\,Дж \cdot кг^{-1} \cdot K^{-1}$ – питома газова стала для сухого повітря.

Використавши другий закон Ньютона (закон збереження імпульсу) до сил, що діють в атмосфері, отримаємо так зване рівняння руху у векторній формі

$$\frac{dV}{dt} = -\frac{1}{\rho} grad\ p - 2\Omega \times V + F_g + \frac{1}{\rho} F_f, \tag{2}$$

де V – вектор швидкості в неінерційній системі координат, що здійснює добове обертання разом із Землею,

F_g – сила тяжіння,

F_f – щільність сили тертя ($H/м^3$),

Ω – вектор кутової швидкості обертання Землі.

Сила F_f, що відповідає одиничному об'єму (далі просто сила), складається із двох компонентів: сила внутрішнього тертя повітря (завдяки власній в'язкості та турбулентності) та сили тертя із поверхнею планети.

Розглянемо рівняння руху (2) у сферичній системі координат (λ, φ, z).

Рівняння, спроектовані на локальні координатні напрямки, набирають вигляд:

$$\begin{aligned}
&\frac{\partial u}{\partial t} + \frac{u}{a\cos\varphi}\frac{\partial u}{\partial \lambda} + \frac{v}{a}\frac{\partial u}{\partial \varphi} + w\frac{\partial u}{\partial z} = -\frac{1}{\rho a\cos\varphi}\frac{\partial p}{\partial \lambda} + \\
&+ \frac{1}{\rho}\frac{\partial}{\partial \lambda}\left(\frac{\rho\mu_{\lambda\varphi}}{(a\cos\varphi)^2}\frac{\partial u}{\partial \lambda}\right) + \frac{1}{\rho}\frac{\partial}{\partial \varphi}\left(\frac{\rho\mu_{\lambda\varphi}}{a^2}\frac{\partial u}{\partial \varphi}\right) + \frac{1}{\rho}\frac{\partial}{\partial z}\left(\rho\mu_z\frac{\partial u}{\partial z}\right) + \\
&+ \left(v\sin\varphi - w\cos\varphi\right)\left(2\omega + \frac{u}{a\cos\varphi}\right),
\end{aligned} \tag{3}$$

$$\begin{aligned}
&\frac{\partial v}{\partial t} + \frac{u}{a\cos\varphi}\frac{\partial v}{\partial \lambda} + \frac{v}{a}\frac{\partial v}{\partial \varphi} + w\frac{\partial v}{\partial z} = -\frac{1}{\rho a}\frac{\partial p}{\partial \varphi} + \\
&+ \frac{1}{\rho}\frac{\partial}{\partial \lambda}\left(\frac{\rho\mu_{\lambda\varphi}}{(a\cos\varphi)^2}\frac{\partial v}{\partial \lambda}\right) + \frac{1}{\rho}\frac{\partial}{\partial \varphi}\left(\frac{\rho\mu_{\lambda\varphi}}{a^2}\frac{\partial v}{\partial \varphi}\right) + \frac{1}{\rho}\frac{\partial}{\partial z}\left(\rho\mu_z\frac{\partial v}{\partial z}\right) - \\
&- u\left(2\omega + \frac{u}{a\cos\varphi}\right)\sin\varphi - \frac{wv}{a}.
\end{aligned} \tag{4}$$

Для атмосфери збереження маси (рівняння нерозривності) записується у звичайній формі

$$\frac{1}{\rho}\frac{d\rho}{dt} + div\,V = 0. \tag{5}$$

Перший закон термодинаміки (рівняння збереження енергії). Тепло δ, що надійшло до одиниці маси повітря, витрачається на зміну внутрішньої енергії частинки та на роботу розширення [8, 9], тобто

$$c_p \frac{dT}{dt} - \frac{1}{\rho}\frac{dp}{dt} = \delta, \tag{6}$$

де $c_V = 717\,Дж \cdot кг^{-1} \cdot K^{-1}$ – питома теплоємність сухого повітря за сталого об'єму,

$c_p = 1004\,Дж \cdot кг^{-1} \cdot K^{-1}$ – питома теплоємність сухого повітря за сталого тиску.

У сферичній системі координат після спрощень остаточний вигляд рівняння (6) буде наступним:

$$\frac{dT}{dt} = \frac{1}{\rho}\frac{\partial}{\partial\lambda}\left(\frac{\rho\mu_{\lambda\varphi}}{(a\cos\varphi)^2}\frac{\partial T}{\partial\lambda}\right) + \frac{1}{\rho}\frac{\partial}{\partial\varphi}\left(\frac{\rho\mu_{\lambda\varphi}}{a^2}\frac{\partial T}{\partial\varphi}\right) + \frac{1}{\rho}\frac{\partial}{\partial z}\left(\rho\mu_z\frac{\partial T}{\partial z}\right). \tag{7}$$

Процеси масштабу, який є меншим за розмір просторової сітки, прийнято параметризувати. Параметризація турбулентності – це встановлення зв'язку турбулентних обмінів, що призводять до виникнення сили внутрішнього тертя, із середніми значеннями метеовеличин. Оберемо для $\mu_{\lambda\varphi}$ та μ_z найпростіші моделі, що ґрунтуються на емпіричних закономірностях теорії турбулентності. При цьому виберемо наступні параметризації турбулентності:

коефіцієнт горизонтального турбулентного обміну рівний (формула Смагоринського) [8, 9]

$$\mu_{\lambda\varphi} = \sigma\left(k_0 + 0.08a\left(\Delta\lambda^2\cos^2\varphi + \Delta\varphi^2\right)\sqrt{D_1^2 + D_2^2}\right), \tag{8}$$

де σ – параметр, що варіює рівень дисипації в моделі; $k_0 = 5\cdot10^4\,м^2/с$ – стала величина; $\Delta\lambda, \Delta\varphi$ – параметри скінченно-різницевої сітки;

$$D_1 = \frac{\partial u}{\partial(\lambda\cos\varphi)} - \frac{\partial v}{\partial\varphi} - vtg\varphi, \qquad D_2 = \frac{\partial v}{\partial(\lambda\cos\varphi)} + \frac{\partial u}{\partial\varphi} + utg\varphi;$$

коефіцієнт вертикального турбулентного обміну за гіпотезою Прандтля [10] становить

$$\mu_z = (b\chi)^2\sqrt{\left(\frac{\partial u}{\partial z}\right)^2 + \left(\frac{\partial v}{\partial z}\right)^2}, \tag{9}$$

де $b = 1500\,м$ – висота граничного шару атмосфери, $\chi = 0.4$ – стала Кармана.

Таким чином, побудована спрощена модель циркуляції вільної атмосфери для макромасштабних процесів складається із рівнянь руху (3), (4); рівняння для вертикальної складової руху (5); рівняння стану (1) для обчислення густини та рівняння теплообміну (7). Горизонтальні та вертикальні коефіцієнти турбулентного обміну визначаються за формулами (8) та (9) відповідно.

Початково-крайові умови моделі

Для постановки початкових та крайових умов моделі використовувалася просторово-часова інтерполяція даних із електронного архіву полів метеорологічних величин.

Паралельний алгоритм

Паралельна реалізація алгоритму розв'язання моделі поєднує два різних типи розщеплення – геометричне та операторне. Розглянемо детальніше особливості цих розщеплень, а також класифікуємо автотюнер, який буде виконувати їх оптимізацію.

Геометричне розщеплення полягає в геометричній декомпозиції області обчислень й дозволяє виконувати обчислення паралельно для кожної під-області. Ця декомпозиція є *статичною* у сенсі того, що залежить від розмірності вхідних даних, а не їх якісних характеристик, тому оптимізацію, що полягає в пошуку оптимального розміру розбиття на під-області, варто виконувати до етапу безпосередніх обчислень, і обране

розбиття буде оптимальним для будь-яких вхідних даних однакової розмірності. Така оптимізація підпадає під категорію автотюнерів з *емпіричним керуванням*.

Операторне розщеплення, а саме: модифіковане адитивно-усереднене розщеплення [7] на відміну від геометричного має вплив на якість отриманих результатів обчислень й залежить від вхідних даних й поточного стану на кожному кроці обчислень (детальніше розглядається в наступному розділі), що робить неможливим оптимізацію цього розщеплення емпіричним автотюнером. Цю задачу буде виконувати автотюнер з *визначеним керуванням*. Розглянемо кожен з описаних підходів.

Модифіковане адитивно-усереднене розщеплення

Розглянемо нестаціонарну крайову задачу із крайовими умовами першого роду

$$\frac{\partial u}{\partial t}+\Lambda u=f\ ,\quad x\in\Omega/\Gamma\quad t\in(0;T]\ , \tag{10}$$

$$u(0,x)=u_0(x),\ x\in\Omega\ , \tag{11}$$

$$u(t,x)=u_\Gamma(t),\ x\in\Gamma=\partial\Omega\quad t\in[0;T]\ , \tag{12}$$

де $\Lambda=\sum_{k=1}^{L}\Lambda_k$ – просторовий оператор, який подається через суму простіших операторів, $f=\sum_{k=1}^{L}f_k$.

Згідно [7] модифікація полягає у введенні незалежного параметра m таким чином: часовий проміжок $[0;T]$ розбивається на відрізки довжини $m\tau$, де τ – часовий крок обчислень. Тоді задача (10) – (12) зводиться до розв'язання низки підзадач (13) та усереднення їх результатів (14) через кожні m кроків

$$\frac{1}{L}\frac{\partial u_\ell}{\partial t}+\Lambda_\ell u_\ell=f_\ell, t\in[qm\tau;(q+1)m\tau]\ ,\ \ell=\overline{1,L}\ , \tag{13}$$

$$u_\ell(qm\tau,x)=y(qm\tau,x),\quad y(0,x)=u_0(x)\ ,$$

$$y((q+1)m\tau,x)=\frac{1}{L}\sum_{\ell=1}^{L}u_\ell((q+1)m\tau,x), \tag{14}$$

де $q=0,...,\lfloor T/m\tau\rfloor$.

Паралельне виконання етапу (13) можливе для всіх L під-задач, оскільки обмін інформацією між ними здійснюється на етапі (14). Тому збільшення значення параметра m призводить до прискорення виконання обчислень, з одного боку, й до збільшення похибки результату, з іншого боку. Величина похибки залежить від темпу росту значення обчислюваних фізичних величин упродовж m-циклу, тобто від обраного часового кроку обчислень τ й від вхідних даних, що надає задачі підбору оптимального значення параметра m динамічний характер і робить її розв'язання можливим тільки за використання автотюнера з *визначеним керуванням*. Розглянемо алгоритм такого керування:

1. Задаємо початкове значення параметра $m=m_0$ у межах $1\le m\le 16$.
2. Виконуємо обчислення на часовому проміжку в $m\tau$ – етапи (13, 14).
3. Обчислюємо чебишевську норму $\varepsilon_{\tau+m}=\dfrac{\|\,y^{\tau+m}-y^{\tau}\,\|}{\left\|y^{\tau}\right\|}$ для вектора швидкості, температури та густини повітря ($\varepsilon_{V^{\tau+m}}$, $\varepsilon_{T^{\tau+m}}$ та $\varepsilon_{\rho^{\tau+m}}$). Нове значення параметра m оберемо наступним чином:

   $m=m_0\times 2$, якщо $\varepsilon_{V^{\tau+m}}\le\dfrac{\varepsilon_V}{2}\ \&\ \varepsilon_{T^{\tau+m}}\le\dfrac{\varepsilon_T}{2}\ \&\ \varepsilon_{\rho^{\tau+m}}\le\dfrac{\varepsilon_\rho}{2}$,

   $m=\dfrac{m_0}{2}$, якщо $\varepsilon_{V^{\tau+m}}\ge\varepsilon_V\ \|\ \varepsilon_{T^{\tau+m}}\ge\varepsilon_T\ \|\ \varepsilon_{\rho^{\tau+m}}\ge\varepsilon_\rho$ (будь-яка з норм перевищує задану межу),

   інакше $m=m_0$, де $\varepsilon_V,\varepsilon_T,\varepsilon_\rho\in[0;1]$ – задана межа відносної похибки на одному головному m-кроці.
4. Якщо $m=m_0$ – виконуємо обчислення з такою величиною m для наступних 3-х годин обчислень. Якщо $m\notin[1;16]$ – обираємо найближче граничне значення й переходимо до п. 2.

Розглянемо *асимптотичну складність* запропонованого алгоритму керування значенням m-параметра. Нехай ми розв'язуємо задачу на вхідних даних, що задають куб розмірністю $n\times n\times l$, де n – кількість вузлів горизонтальної сітки, а l – кількість вузлів вертикальної сітки. Обчислення всіх норм має складність $O(n^2l)$ і виконується в кінці кожного m-циклу. На кожному часовому кроці кожна підзадача (13) виконує $O(n^4l)$ обчислень, що дає $O(mn^4l)$ для всього m-циклу. Беручи до уваги п. 4 алгоритму, а також те, що підрахунок

норм виконується після етапу (14) й не потребує додаткової синхронізації, можна вважати, що обране визначене керування не впливає значною мірою на загальну складність обчислень.

Покоординатна декомпозиція області обчислень

Надамо означення використаної декомпозиції для p-мірного гіперкубу $\Omega = [0;1]^p$. Ця декомпозиція задається наступним чином: $\Omega^{(k)} = \left\{ \Omega_i^{(k)} : \bigcup_{i=1}^{s_k} \Omega_i^{(k)} = \Omega \right\}$ за умови, що принаймні за k-м координатним напрямком область Ω залишалася цілою. Тобто, є дві умови, які має задовольняти наша декомпозиція $\Omega^{(k)}$:

1) $diam\, \ell_k = 1$ для довільного відрізка ℓ_k із $\Omega_i^{(k)} \in \Omega^{(k)}$, $i = 1,...,s_k$, що паралельний вісі Ox_k;

2) також можуть існувати інші координатні напрямки з властивістю (п. 1).

Оскільки наша область обчислень задається кубом, то покажемо на рис.1 дві множини покоординатних декомпозицій за всіма напрямками для випадку $\Omega = [0;1]^3$.

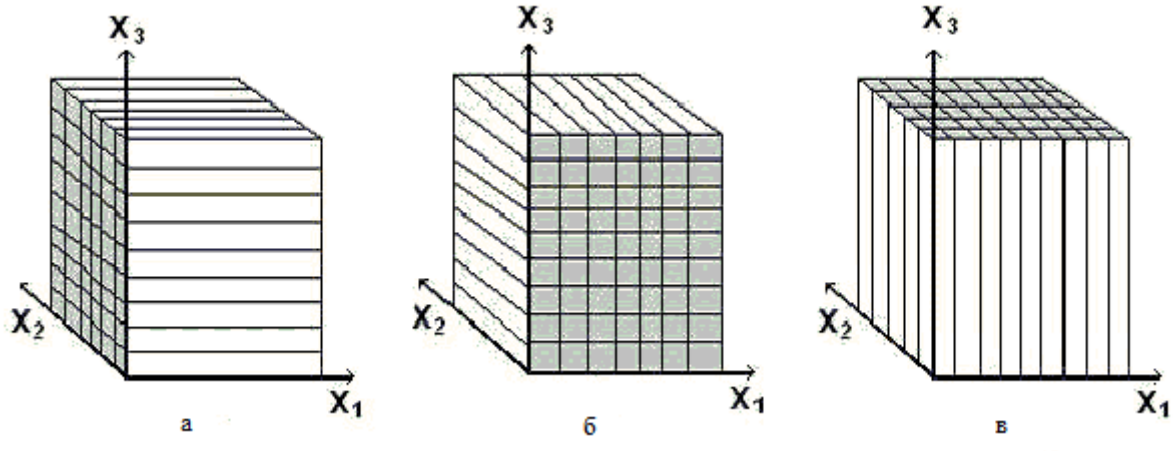

Рис. 1. Множина індивідуальних покоординатних декомпозицій:
а – для напрямку x_1; б – для напрямку x_2; в – для напрямку x_3

Декомпозиції із $\left\{\Omega^{(3)} : k = 1,2,3\right\}$, які показані на рис. 1, задовольняють лише п. 1, тому кожна покоординатна декомпозиція $\Omega^{(k)}$ відповідає лише одному просторовому напрямку.

На рис. 2 кожна із двох декомпозицій задовольняє п. 1 та п. 2. Кожній декомпозиції відповідають два напрямки; напрямку Ox_2 відповідають обидві декомпозиції.

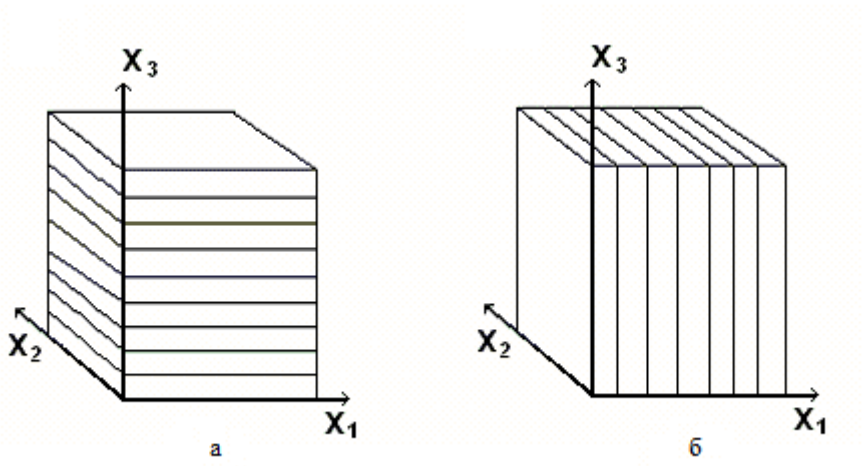

Рис. 2. Обрана геометрична декомпозиція

Застосування описаної декомпозиції до вхідних даних породжує $S \times p \times e$ незалежних під-задач, де S – кількість під-областей декомпозиції, p – розмірність простору ($p = 3$), а e – кількість еволюційних рівнянь моделі (у нашій моделі їх три). Загальна ефективність обчислень залежить від величини S. При малих значеннях S кількість операцій, що виконуються кожною під-задачею, досить велика (як вже зазначалося,

асимптотична складність має порядок $O(\frac{n^4 l}{S^5})$, де n – кількість вузлів горизонтальної сітки, а l – кількість вузлів вертикальної сітки), це компенсує затрати на синхронізацію в кінці кожного m-циклу. З іншого боку, зважаючи на таку характерну особливість обраної задачі, як малий розмір використовуваних даних, цілком можливим є розбиття, при якому дані кожної під-задачі будуть повністю поміщатися в кеш процесора, що призведе до суперприскорення алгоритму [11, 12]. Оскільки оптимальне значення параметра S буде різним для різних обчислювальних платформ, підбір оптимальної розмірності декомпозиції буде виконувати автотюнер з емпіричним керуванням.

Як вже було зазначено, динамічний підбір параметра модифіковано адитивно-усередненого розщеплення виконує автотюнер з *визначеним керуванням*, який є частиною застосунка. Наведемо опис автотюнера для підбору розмірності оптимальної декомпозиції області обчислень.

Паралельна реалізація задачі здійснювалася засобами бібліотеки OpenMP й отримувала всі параметри для виконання через командний рядок. Задачею автотюнера є пошук такої конфігурації цих параметрів, за якої обчислення будуть виконуватися максимально швидко (слід зауважити, що також виконувався пошук конфігурації, за якої програма виконується максимально ефективно, тобто показує максимальне мультипроцесорне прискорення). Оскільки програма приймає усі параметри, що впливають на її швидкодію, з командного рядка – необхідність у трансформації вихідного коду програми й її перекомпіляції у процесі тюнінгу відсутня, що значно спрощує структуру автотюнера. Наступна діаграма описує його роботу (рис. 3).

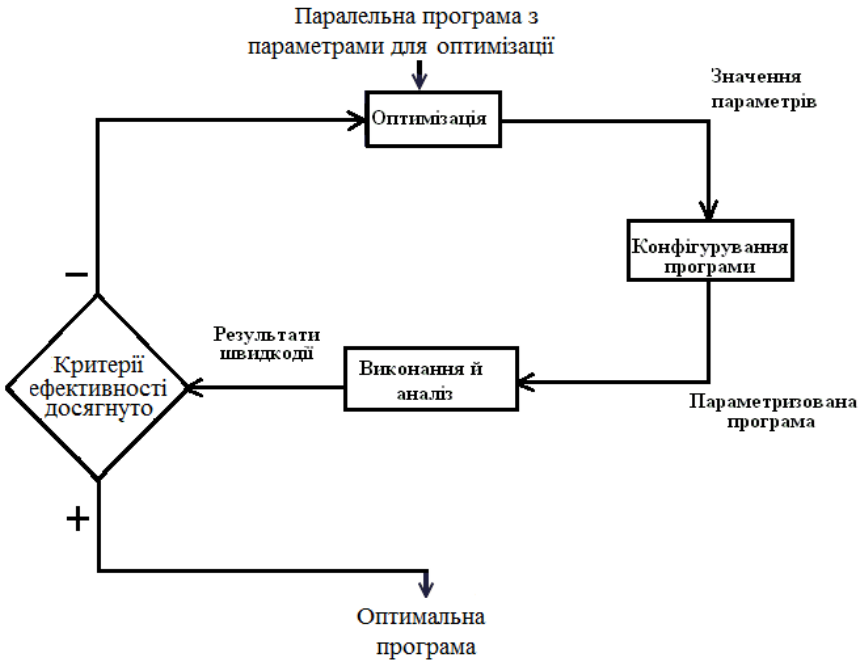

Рис. 3. Діаграма роботи автотюнера

Автотюнер є окремим застосунком, створеним засобами мови Java, і приймає на вхід межі пошуку величини декомпозиції S. Нехай ми можемо використовувати для обчислень n процесорів, тоді доцільно визначити область пошуку параметра як $1 \le S \le 2 \times n$. Для оцінки швидкодії в програму було додано замір часу виконання обчислень з використанням бібліотечної функції omp_get_wtime(), результати якого виводилися у текстовий файл, який автотюнер аналізував по закінченні обчислень. Зберігалися значення, за яких часові затрати на виконання обчислень були мінімальними, а також коефіцієнт ефективності обчислень $E(k) = \frac{W(k)}{N_{проц}}$ був максимальним, де $W(k) = \frac{T_1}{T_k}$ – мультипроцесорне прискорення, T_1 й T_k – час обчислень за використання одного й k процесорів відповідно.

Зауважимо, що відмова від виконання автотюнером трансформацій вихідного коду певною мірою обмежує його можливості. Наприклад, без трансформації коду неможлива зміна внутрішнього представлення даних у програмі, а також напрямків їх обходу, а експериментування з ними показало різницю в загальній швидкодії обчислень приблизно в 15%. Проте навіть у такому спрощеному вигляді автотюнер показав цілком задовільні результати, що розглядатимуться далі.

Результати чисельного експерименту

Чисельний експеримент проводився на гомогенній ЕОМ з 24 процесорами Dual Core Intel Itanium 2 Series 9000 із частотою 1.6 ГГц та архітектурою IA64. Система доступу до пам'яті NUMAlink 4, топологія доступу fat-tree, загальний обсяг оперативної пам'яті становив 96 ГБ.

Обчислення виконувалися з наступними параметрами:

- величина часового кроку $\Delta t = 10$ сек;
- обмеження для m-параметра: $1 \le m \le 16$;
- обчислення проводилися на сітці розмірністю $180 \times 180 \times 11$.

У результаті чисельного експерименту знайдена декомпозиція $S = 4$ (на рис. 4 – 6 позначена як '*'), при якій програма показала мультипроцесорне прискорення $W(24) = 18.36$ й загальну ефективність $E(24) = 0.76$, що є гарним показником.

Слід зауважити, що оскільки розглянута модель є тривимірною і описується трьома еволюційними рівняннями, то $S = 4$ означає, що після декомпозиції ми маємо $4 \times 3 \times 3 = 36$ під-областей.

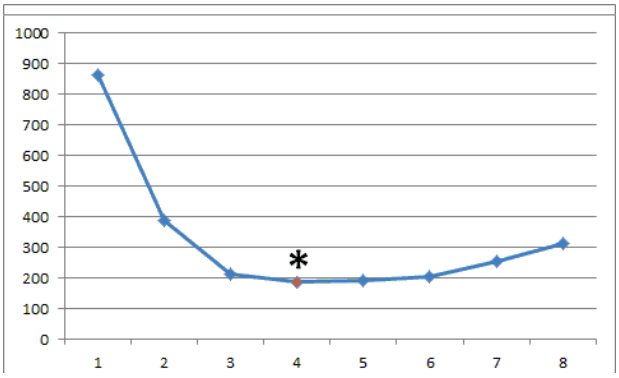

Рис. 4. Залежність часу виконання обчислень від S

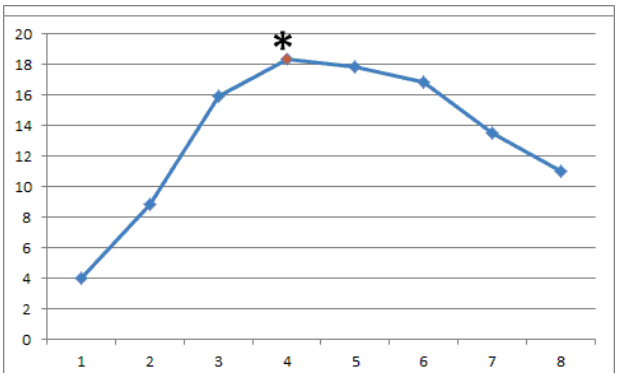

Рис. 5. Залежність мультипроцесорного прискорення від S

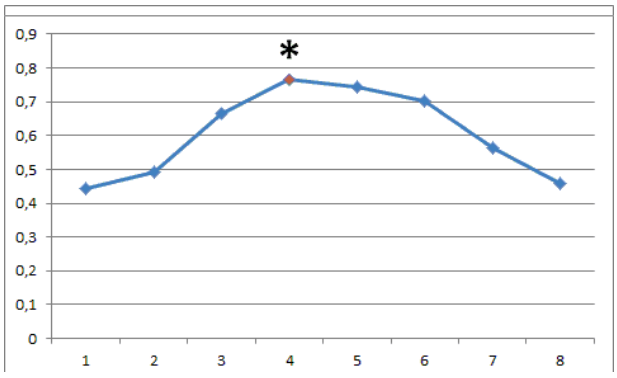

Рис. 6. Залежність ефективності обчислень $E(k)$ від S

Висновки

У роботі широко розглянуто поняття автотюнінгу, його класифікацію та клас вирішуваних ним задач. На прикладі моделі для короткочасного метеорологічного прогнозування було проілюстровано застосування автотюнінгу для оптимізації обчислень прикладних наукових задач високої складності.

Обрана модель є досить складною, оскільки її оптимізація не обмежується *евристичним пошуком* оптимальної конфігурації на етапі інсталяції й вимагає *визначеного керування* під час виконання обчислень. Основою паралельного алгоритму є операторне розщеплення, яке на відміну від геометричного впливає не тільки на швидкодію, але й на точність отриманого розв'язку, що вимагає компромісу. Необхідність компромісу підкріплюється й характером розв'язуваної задачі тому що короткочасне прогнозування вимагає максимально швидких й точних результатів. Тому високі показники практичного експерименту наглядно демонструють можливості автотюнінгу й підтверджують ефективність його застосування до задач високої складності.

Також був розглянутий підхід до покращення отриманих результатів, а саме: розробку фреймоворку для автотюнінгу з підтримкою *декларативного опису* виконуваних *трансформацій вихідного коду* програм, який є гнучкішим і загальнішим рішенням, а також позбавлений обмежень описаного в статті автотюнера.

1. *Schaefer C.A., Pankratius V., and Tichy W. F.* Atune-IL: An instrumentation language for auto-tuning parallel applications // Euro-Par '09 Proceedings of the 15th International Euro-Par Conference on Parallel Processing Springer-Verlag Berlin, Heidelberg 2009.
2. *Schaefer C.A.* Reducing Search Space of Auto-Tuners Using Parallel Patterns // IWMSE '09 Proceedings of the 2009 ICSE Workshop on Multicore Software Engineering IEEE Computer Society Washington, DC, USA 2009.
3. *Kamil S., Chan C., Williams S., Oliker L., Shalf J., Howison M., Bethel E.W., Prabhat* A Generalized Framework for Auto-tuning Stencil Computations. Proceedings of the Cray User Group Conference, 2009.
4. *Mishra S.K.* Global Optimization by Particle Swarm Method:A Fortran Program, 2006.
5. *Chen H., Zhang W., Jiang G.*, Experience Transfer for the Configuration Tuning in Large Scale Computing Systems // IEEE Transactions on Knowledge and Data Engineering archive, Vol. 23, Issue 3, March 2011.
6. *Chung I., Hollingsworth J. K.* Using Information from Prior Runs to Improve Automated Tuning Systems. SC '04 Proceedings of the 2004 ACM/IEEE conference on Supercomputing // IEEE Computer Society Washington, DC, USA 2004.
7. *Черниш Р.І.* Модифіковане адитивно-усереднене розщеплення, його паралельна реалізація та застосування до задач метеорології: автореф. дис. канд. фіз.-мат. наук // КНУ ім. Тараса. Шевченка. К., – 2010. – 20 с.
8. *Матвеев Л.Т.* Теория общей циркуляции атмосферы и климата Земли.– Л. : Гидрометеоиздат, 1991. – 295 с.
9. *Атмосфера* : справочник / [научн. редкол.: Ю. С. Седунов и др.] – Л. : Гидрометеоиздат, 1991. – 601 с.
10. *Марчук Г.И., Дымников В.П., Залесный В.Б.*, Математическое моделирование общей циркуляции атмосферы и океана. – Л. : Гидрометеоиздат, 1984. – 320 с.
11. *Frigo M., Leiserson C.E., Prokop H., Ramachandran S.* Cache-Oblivious Algorithms (Extended Abstract).Proc. 40th Annual Symposium on Foundations of Computer Science, 1999.
12. *Prokop H.* Cache-Oblivious Algorithms. Department of Electrical Engineering and Computer Science, 1999.
13. *Naono K., Teranishi K., Cavazos J., Suda R.* Software Automatic Tuning From Concepts to State-of-the-Art Results, Springer; 1st edition, September 21, 2010.