УДК 681.3

А.Ю. Дорошенко, О.П. Ігнатенко, П.А. Іваненко

ПРО ОДНУ МОДЕЛЬ ОПТИМАЛЬНОГО РОЗПОДІЛУ РЕСУРСІВ У БАГАТОПРОЦЕСОРНИХ СЕРЕДОВИЩАХ

Розглядається модель керування ресурсами в однорідному багатопроцесорному середовищі. Запропоновано підхід на основі потокових моделей, який дозволяє отримати оптимальне у сенсі швидкодії керування. Отримано аналітичний вид часу закінчення роботи в залежності від параметрів розпаралелювання. Результати проілюстровані на відомому прикладі множення матриць. Виконано експериментальне підтвердження теоретичних моделей.

Вступ

Сучасні задачі науки і техніки часто вимагають значних обсягів обчислювальних ресурсів, тому важливою задачею є підвищення ефективності виконання програм у багатопроцесорних середовищах. Керування виконанням заданої програми залежить від різних факторів: архітектури системи, операційної системи та організації виконання (розпаралелювання програми). Архітектура і операційна система, як правило, фіксовані, тому основним напрямком керування є організація виконання програми. Для забезпечення ефективного використання ресурсів багатопроцесорної системи необхідно розв'язати дві основні задачі [1].

По-перше, обчислювальне навантаження програми має бути рівномірно розділене між процесорами з мінімально можливими обмінами між ними. Це – *задача розподілу*.

По-друге, потрібно поставити у відповідність кожному завданню час початку виконання і процесор, на якому це завдання буде виконуватись. При цьому бажано, щоб процесори були завантажені неперервно. Це – *задача планування*. Обидві ці задачі *NP*-складні.

Загальна задача керування є оптимізаційною, де як міра ефективності може виступати загальний час виконання, вартість використання, усереднений час виконання кожного завдання (час очікування + час виконання + час передачі користувачу). Яскравим прикладом розподіленої обчислювальної системи є Грід. Грід [2] це конгломерат обчислювальних ресурсів, з'єднаних у мережу, який використовується для розв'язання ресурсоємних задач. Характерною особливістю Грід є значна розподіленість і неоднорідність. Окремі ресурси можуть бути поєднані напряму, через локальну мережу або Інтернет, що спричиняє зміни у доступності та вартості використання різних частин системи.

Існує дві основні категорії схем розподілу (або керування) – статичні і динамічні [3]. Статичні схеми керування використовують накопичені статистичні дані про роботу системи і не враховують поточний стан. Такий підхід дозволяє побудувати швидку і просту програму керування. Динамічні схеми використовують для прийняття рішень поточний стан системи (розміри черг, завантаженість і доступність вузлів, параметри вхідного потоку завдань). У рамках цього підходу окремі вузли мережі можуть обмінюватися інформацією і намагаються збалансувати навантаження шляхом перерозподілу ресурсів між вузлами. Хоча це і вимагає витрат певної частини ресурсів системи на забезпечення обміну повідомленнями і обчислення керування, динамічні схеми, взагалі кажучи, гарантують значно кращу ефективність роботи.

Завдання, що виконуються в розподіленій системі можуть бути як однорідні так і розділені на класи. Таке розділення може здійснюватися за ознакою користувача (всі завдання одного користувача виділяють у окремий клас), за розміром або пріоритетом. Критерії ефективності схеми керування розділяють на [4].

- Загально-оптимальні. Всі завдання належать одному класу і мінімізується загальна характеристика.
- Індивідуально-оптимальні. Мінімізується час виконання кожного окремого завдання.
- Оптимальні за типами. Виділяється скінченна кількість типів завдань, схема мінімізує час виконання завдань кожного типу.

Задача статичного керування ресурсами для одного і багатьох класів завдань досліджена досить добре. Більшість робіт розглядають ситуацію з точки зору глобального підходу, коли основна увага зосереджена на мінімізації очікуваного часу відклику системи у цілому. При цьому розглядались різні конфігурації мережі як для нелінійних [5 – 6], так і для лінійних [7] моделей. У цих роботах вирішується задача мінімізації часу відклику. Дана робота також будує статичну схему керування, яка мінімізує час завершення завдання.

Деякі автори розглядають статичні алгоритми, що забезпечують індивідуально-оптимальний розв'язок [8] та спираються на методи класичної теорії ігор. Теоретико-ігровий підхід застосовувався, зокрема, у роботах [8] (дослідження некооперативних ігор), [9, 10] (кооперативні ігри, рівновага за Нешем), для моделювання динаміки Грід систем [11 – 13].

Динамічне керування, на відміну від статичного, має обчислюватись у реальному часі. Для розподілених систем це означає, що вузли мають обмінюватись інформацією про свій поточний стан. Для великої кількості вузлів такі обміни можуть негативно впливати на роботу всієї системи, тому розглядають окремо централізоване і розподілене керування. У першому випадку виділяється окремий комп'ютер (вузол), відповідальний за відстеження глобального стану системи. На основі зібраної інформації цей керуючий вузол приймає рішення щодо розподілу завдань і ресурсів. При побудові централізованих схем керування застосовують теорію черг та інші моделі.

Розподілене або децентралізоване керування характерне для систем зі значною кількістю вузлів. У цьому випадку кожен окремий вузол приймає рішення про розподіл ресурсів на основі доступної йому локальної інформації (яка може бути неповною). Тому більшість розподілених схем є субоптимальними або евристичними саме через неможливість отримати повну й достовірну інформацію про поточний стан системи. Розподілені динамічні схеми досліджувались у роботах [14]. Великий клас алгоритмів був отриманий з використанням ідей штучного інтелекту [15] та генетичних алгоритмів [16].

2. Задача розподілу ресурсів при виконанні паралельних обчислень

Задача розподілу ресурсів у неоднорідних багатопроцесорних системах при здійсненні паралельних обчислень є складною проблемою, оскільки включає у себе розподіл за двома вимірами – час і ресурси, у два етапи – маршрутизація (передача пакетів на вузли) і планування (надання ресурсів вузлам для здійснення обчислень).

Існують різні підходи до розв'язання цієї задачі. Найбільш відомі з них: спільна черга, кластерізація і дворівневий розподіл.

Спільна черга. Можливо найпростішим способом організації паралельного розподілу ресурсів є спільна черга (Global queue). У цьому випадку на кожному вузлі записується екземпляр програми, яка обмінюється даними зі спільною структурою в якій зберігається інформація про всі поточні завдання. Система переносить першу задачу з черги на вільний процесор для виконання. Основною перевагою спільної черги є простота й автономність розподілу. Крім цього, процесор завантажується роботою відразу після завершення і працює поки в черзі є завдання.

Однак недоліки схеми також значні. По-перше, при зростанні кількості процесорів спільна черга починає негативно впливати на роботу системи. По-друге, при завантаженні нового завдання втрачаються всі дані локальної пам'яті (при скоординованому плануванні, можливо, деякі дані могли б бути використані вдруге). І, наре-

шті, останнє – дана схема не враховує необхідності взаємодії різних завдань під час виконання.

Кластерізація. Іншим підходом є розділення процесорів на окремі незалежні множини (кластери) і виконання кожної роботи на окремому кластері. Існують різні способи розділення: фіксоване, за вимогою користувача, адаптивне і динамічне. Розділення за запитом користувача є одним з найбільш популярних. Перевагою підходу є врахування потреб користувача в першу чергу і ексклюзивний доступ до ресурсів і пам'яті вузлів, виділених для задачі. Звичайно, розділення за запитом має свої недоліки. Причина їх виникнення полягає у можливій невідповідності запитів користувачів і наявних ресурсів. У зв'язку з цим виникають дві проблеми:

- перерозподіл вільних вузлів, якщо їх ресурсів недостатньо для виконання поточних завдань;
- виконання великого завдання, що вимагає значних ресурсів. Така задача може очікувати досить довго поки не звільниться достатня кількість ресурсів.

Динамічний дворівневий розподіл. Одним з способів зменшення часу очікування завдань у черзі є попередження монополізації великої кількості процесорів одним завданням. Ефективним способом організації обчислень є використання адаптивних схем розподілу ресурсів. Однак, для застосування ідеї адаптивного розподілу необхідно розв'язати проблему керування завантаженням вузла під час виконання завдання. Кількість процесорів, виділену для завдання, змінювати під час виконання не можна, тому для організації динамічного розділення виділяють два рівня розміщення задачі. Операційна система виділяє вузли для обчислень, а спеціальне програмне забезпечення розміщує обчислення в залежності від наявної кількості ресурсів.

Динамічний дворівневий розподіл – схема, що широко застосовується і, як показано, має кращі характеристики ніж інші схеми.

Це досягається завдяки тому, що

1. Ресурси не витрачаються на перерозподіл вільних вузлів.
2. Непотрібно здійснювати синхронізацію на глобальному рівні.
3. Рівень розпаралелювання завдань автоматично змінюється за умов перевантаження, що покращує ефективність використання системи.

3. Модель системи

Будемо розглядати систему з P незалежних процесорів, які утворюють кластер. Існує нульовий вузол, який ми позначимо p_0 – маршрутизатор, на який потрапляють завдання. Вузол p_0 здійснює пересилку завдань на процесори. Потужність процесорів описується функціями $\mu_i u_i(t)$, $i = 1,..,P$, де $u_i(t) \in [0,1]$ – нормалізоване керування, μ_i – параметр потужності даного процесора.

У відповідності до підходу, викладеному в роботі [17] будемо моделювати процес планування за допомогою потокової моделі, вважаючи, що кількість обчислень, які необхідно здійснити на момент часу t для виконання заданих робіт може бути представлена неперервною функцією часу. В певному розумінні, даний підхід є двоїстим до ідеї, викладеної у [1], де неперервними функціями описується сумарна потужність процесорів, що розв'язують задачу.

Отже, позначимо $q_i(t)$, $i = 1,..,P$ – кількість обчислень, які необхідно виконати p_i процесору для завершення поточного завдання. Тоді в початковий момент часу $q_i(0) = c_i > 0$. Причому під час виконання на процесори можуть надходити нові завдання.

Якщо визначена функція керування $u_i(t)$ і нових надходжень немає, то

$$\dot{q}_i = -\mu_i u_i(t),\ i = 1,..,P.$$

Тобто вважається, що швидкість обчислення залежить тільки від поточної потужності процесора.

Опишемо тепер процес маршрутизації завдань. Нехай $\lambda(t)$ – функція, що описує завантаження завдань на вузол p_0.

Наприклад, така функція може мати вигляд подібний до зображеного на рис. 1.

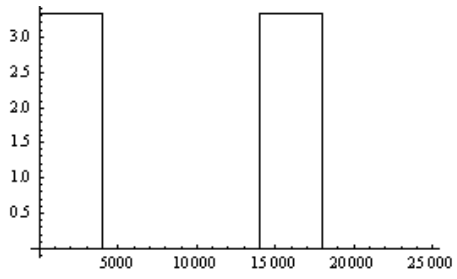

Рис. 1. Приклад функції $\lambda(t)$

При цьому природно обмежити $\lambda(t)$ наступним чином:

$$\lambda(t) \in [0, \lambda^{\max}],$$
$$\int_0^{\infty} \lambda(t)dt = \lambda^{\text{int}}.$$

Тоді динаміка системи буде описуватися потоковою моделлю:

$$\begin{aligned} \dot{q}_0(t) &= \lambda(t) - \rho v_1(t) - ... - \rho v_P(t), \\ \dot{q}_1(t) &= f_c(v_1(t)) - \mu_1 u_1(t), \\ &\vdots \\ \dot{q}_P(t) &= f_c(v_P(t)) - \mu_P u_P(t), \end{aligned} \qquad (1)$$

або у скороченому вигляді:

$$\frac{dq(t)}{dt} = \tilde{f}(\lambda(t), v(t), u(t)).$$

При цьому:

$v(t) = (v_1(t),...,v_P(t)) \in V$ – функція маршрутизації,

$u(t) = (u_1(t),...,u_P(t)) \in U$ – функція керування виконанням,

$\lambda(t)$ – функція завантаження вхідних завдань,

$f_c(\cdot)$ буде введена далі.

Множини керування $U \subset R^P$, $V \subset R^P$ описують обмеження на ресурси. Простим і поширеним випадком [18] є випадок лінійних обмежень, коли, наприклад, U – поліедральна множина, задана співвідношенням:

$$U = \left\{ u \in R^P : Cu \le \overline{1}, u_i \ge 0, i \in I \right\},$$

де C – дійсна матриця, $\overline{1}$ – вектор з одиниць і нерівність виконується по рядкам. Кожен рядок матриці C відповідає певному ресурсу. В рамках задачі множення матриць і кластерної архітектури приймемо, що $C_1 = C_2 = I$, тобто

$$U = \left\{ u \in R^P : u_i \le 1, u_i \ge 0, i \in I \right\},$$
$$V = \left\{ v \in R^P : v_i \le 1, v_i \ge 0, i \in I \right\}.$$

У рамках даної моделі враховується відмінність між передачею даних для обчислень на процесорі p_i і самими обчисленнями (існуючі потокові моделі, наприклад, [17] ці процеси не розрізняють). Тим не менше відмінність існує. Наприклад, у задачі множення квадратних матриць розмірності $N \times N$ необхідно передавати на процесори $2N^2$ чисел, при цьому потрібно виконати обсяг обчислень порядку $O(N^3)$. Для врахування цієї нелінійної залежності й вводиться функція $f_c(\cdot) : R \to R$, яка перетворює інтенсивність передачі даних $v_i(t)$ в інтенсивність отримання завдання на обчислення $f_c(v_i(t))$.

Задача оптимального керування. Тепер ми можемо поставити задачу оптимальної організації обчислень. Нехай

$$T = \min\{t \ge 0 : q(t) = 0, \lambda^{\text{int}} - \int_0^t \lambda(\tau)d\tau = 0\},$$

тоді задача зводиться до мінімізації часу за наступних умов:

$$\begin{aligned} &T \to \min, \\ &\frac{dq(t)}{dt} = \tilde{f}(\lambda(t), v(t), u(t)),\ v(t) \in V, \\ &q(t) \ge 0. \end{aligned} \qquad (2)$$

При цьому функція $\lambda(t)$ – фіксований параметр. Задача (2) це задача оптимального керування з нефіксованим часом за наявності фазових обмежень. Для цієї задачі вірна теорема про принцип максимуму, якщо функція $\tilde{f}(\lambda, v, u)$ гладко залежить від параметрів [19]. Оскільки

залежність від λ, u лінійна, то умова виконується якщо $f_c(\cdot)$ гладка функція.

Твердження. Якщо існує оптимальний розв'язок задачі (2), то мінімальний час T досягається для $\hat{\lambda}(t) = \lambda^{\max}$, $t \in \left[0, \frac{\lambda^{\text{int}}}{\lambda^{\max}}\right]$.

Доведення. T не може бути менше за момент часу, коли $\lambda^{\text{int}} - \int_0^t \lambda(\tau)d\tau = 0$. Цей момент мінімальний при вказаному $\hat{\lambda}(t)$. Залишилось показати, що для всіх інших функцій $\lambda(t)$ $\min\{t \geq 0 : q(t) = 0\}$ принаймі не зменшується.

Дійсно, наприклад, для $q_0(t)$ такий момент часу визначається рівнянням:

$$\int_0^t \lambda(\tau)d\tau = \lambda^{\text{int}} = \rho t,$$

за інших $\lambda(t)$ цей момент часу може зрости або залишитися незмінним.

Наслідок. Для оптимального розв'язку задачі (2) функції керування $v(t)$, $u(t)$ – кусково лінійні та мають не більше одного переключення. Це випливає з того, що $\tilde{f}(\cdot)$ не залежить від $q(t)$.

4. Задача множення матриць

4.1. Постановка задачі. Результатом множення матриці A розмірністю $m \times n$ й матриці B розмірності $n \times l$ є матриця C розмірності $m \times l$, кожен елемент якої обчислюється наступним чином:

$$c_{ij} = \sum_{k=0}^{n-1} a_{ik} \times b_{kj}, 0 \leq i < m, 0 \leq j < l.$$

Цей алгоритм вимагає $m \times n \times l$ операцій множення й додавання елементів матриць. При множенні квадратних матриць розмірності $n \times n$ кількість виконаних операцій має порядок $O(n^3)$. Необхідно побудувати паралельний алгоритм, який буде виконувати множення матриць за найкоротший час для довільної архітектури кластера.

4.2. Блочний алгоритм. Для експерименту було обрано наступну паралельну модифікацію послідовного алгоритму:

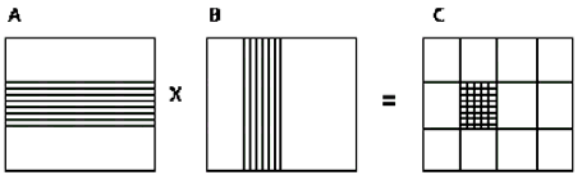

Рис. 2 Схема множення

Якщо ми множимо дві квадратні матриці розмірністю $n \times n$, то результуюча матриця C буде тієї самої розмірності, що й вхідні матриці A і B. Спочатку матриця C розбивається на прямокутні блоки розмірністю $\frac{n}{xCount} \times \frac{n}{yCount}$, де $xCount$ та $yCount$ – кількість блоків по вертикалі й горизонталі. Кожен з цих блоків обчислюється незалежно окремою під-задачею, для чого під-задача потребує блок матриці A розмірністю $n \times \frac{n}{xCount}$ й аналогічний блок матриці B розмірністю $\frac{n}{yCount} \times n$.

Перевагою такого алгоритму є те, що задачу можна розбивати на довільну кількість підзадач, а не тільки на квадратні блоки. Також відсутність будь-яких залежностей між підзадачами дозволяє ефективно виконувати обчислення у випадку коли кількість наявних процесорів значно менша за кількість під-блоків – у такому випадку підзадачі отримують нові блоки «за готовністю».

Недоліком є додаткові витрати пам'яті, які виникають при частковому дублюванні вхідних даних для різних підзадач.

Загальна кількість операцій не відрізняється від послідовного алгоритму й має порядок $O(n^3)$.

Кожна підзадача виконує $\frac{n}{xCount} \times \frac{n}{yCount} \times (2n-1)$ скалярних операцій. Тому час обрахунків кожної підзадачі буде рівним:

$$T(calc) = \frac{n}{xCount} \times \frac{n}{yCount} \times (2n-1) \times \tau .$$

Оцінка комунікаційних затрат для кожної під-задачі буде наступною:

$$TCount = \frac{n^2}{xCount} + \frac{n^2}{yCount} + \frac{n^2}{xCount \cdot yCount},$$

$$T(comm) = \alpha + \frac{\omega \times TCount}{\beta},$$

де α – латентність, β – пропускна здатність мереж, ω – розмір елемента матриці в байтах.

Для реалізації обчислень була обрана схема «постачальник – споживач» в якій підготовкою, розсилкою даних, а також обробкою результатів обчислень займається задача – «постачальник». Обчислення виконуються на «споживачах».

Перевагою такої схеми є відсутність затрат на синхронізацію обчислень й мінімальний час очікування «споживачів», особливо коли кількість під-задач значно більше кількості задіяних процесорів. Недоліком є використання додаткового процесора для «майстра», який не приймає участь в обчисленнях.

4.3. Математична потокова модель. Розглянемо випадок P однакових серверів з окремою пам'яттю та незалежними каналами зв'язку (рис. 3).

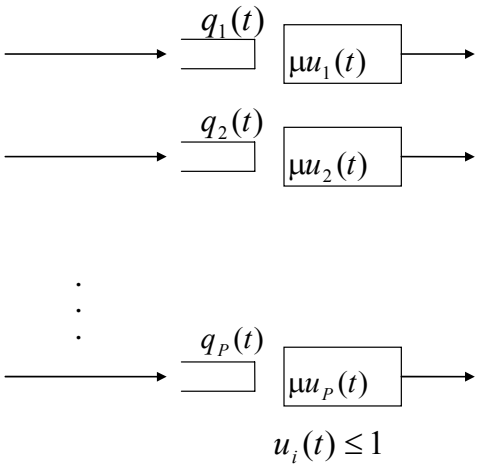

Рис. 3. P однакових серверів з окремою пам'яттю

Обмеження на керування мають вигляд:

$$\sum_i v_i(t) \le 1,\ 0 \le v_i(t) \le 1,\ i = 1,...,P,$$

$$0 \le u_i(t) \le 1,\ i = 1,...,P.$$

Будемо вважати, що $q_R(t_0) = 0$, $q_i(t_0) = 0$, $i = 1,...,P$.

Якщо $\lambda^{\max} < \rho$, тоді визначимо функції маршрутизації

$$v_i(t) = \begin{cases} \dfrac{\lambda(t)}{n\rho}, & \lambda(t) > 0, \\ 0, & \lambda(t) = 0. \end{cases}$$

$$\dot{q}_R(t) = 0,$$
$$q_R(t) = 0.$$

Час завантаження задач

$$T_1 = \frac{\lambda^{\text{int}}}{\lambda^{\max}}.$$

Черги на серверах дорівнюють

$$q_i(t) = \int_0^t f(v(\tau))d\tau - \mu t$$

при керуванні

$$u_i(t) = \begin{cases} 1, & q_i(t) > 0, \\ 0, & q_i(t) = 0. \end{cases}$$

Введемо позначення

$\lambda_f^{\max} = \dfrac{\lambda_f^{\text{int}} \cdot \lambda^{\max}}{\lambda^{\text{int}}}$, де $\lambda_f^{\text{int}} = f_c(\lambda^{\text{int}})$.

Якщо $\lambda_f^{\max} < \mu$, то $T_2 = T_1$, де T_2 – мінімальний час закінчення роботи.

Якщо $\lambda_f^{\max} \ge \mu$, то

$$\mu T_2 = \int_0^{\infty} f(v(\tau))d\tau = \int_0^{T_1} f(v(\tau))d\tau =$$

$$= T_1 \lambda_f^{\max} = \frac{\lambda^{\text{int}}}{\lambda^{\max}} \cdot \frac{\lambda_f^{\text{int}} \lambda^{\max}}{P\lambda^{\text{int}}} = \frac{\lambda_f^{\text{int}}}{P},$$

$$T_2 = \frac{\lambda_f^{\text{int}}}{P\mu}.$$

Максимальна черга при цьому дорівнює

$$q_i^{\max}(t) = \frac{\lambda_f^{\text{int}}}{P} - \mu \frac{\lambda^{\text{int}}}{\lambda^{\max}}.$$

Якщо $\lambda^{\max} \geq \rho$, тоді визначимо функції маршрутизації

$$v_i(t) = \begin{cases} \dfrac{1}{P}, & \lambda(t) > 0, \\ 0, & \lambda(t) = 0. \end{cases}$$

$$q_R(t) = \int_0^t \lambda(\tau) d\tau - \int_0^t \rho \cdot v_1(\tau) d\tau - \ldots - \int_0^t \rho \cdot v_K(\tau) d\tau.$$

Час закінчення завантаження задач

$$T_1 = \frac{\lambda^{\text{int}}}{\rho}.$$

Максимальна черга

$$q_R^{\max} = \lambda^{\text{int}} - \rho \frac{\lambda^{\text{int}}}{\lambda^{\max}}.$$

Черги на серверах дорівнюють

$$q_i(t) = \int_0^t f(v(\tau)) d\tau - \mu t.$$

при керуванні

$$u_i(t) = \begin{cases} 1, & q_i(t) > 0, \\ 0, & q_i(t) = 0. \end{cases}$$

Якщо $\lambda_f^{\max} < \mu$, то $T_2 = T_1$.

Якщо $\lambda_f^{\max} \geq \mu$, то

$$\mu T_2 = \int_0^{\infty} f(v(\tau)) d\tau = \int_0^{T_1} f(v(\tau)) d\tau = T_1 \lambda_f^{\max} = \frac{\lambda^{\text{int}}}{\rho} \cdot \frac{\lambda_f^{\text{int}} \rho}{P \lambda^{\text{int}}} = \frac{\lambda_f^{\text{int}}}{P},$$

$$T_2 = \frac{\lambda_f^{\text{int}}}{P\mu}.$$

Максимальна черга при цьому дорівнює

$$q_i^{\max} = \frac{\lambda_f^{\text{int}}}{n} - \mu \frac{\lambda^{\text{int}}}{\lambda^{\max}}.$$

Таким чином, час закінчення завантаження дорівнює

$$T_1 = \frac{\lambda^{\text{int}}}{\min(\rho, \lambda^{\max})}.$$

Мінімальний час виконання

$$T_2 = \frac{\lambda_f^{\text{int}}}{P \cdot \min\left(\mu, \dfrac{\lambda_f^{\text{int}} \cdot \min(\rho, \lambda^{\max})}{P \cdot \lambda^{\text{int}}}\right)}.$$

Припустимо, що вхідний потік розділено на k частин, і використаємо щойно отриману формулу.

Для реальної системи виконуються співвідношення:

$$\lambda^{\max} > \rho, \ \lambda_f^{\max} > \mu.$$

Якщо $k \leq P$, то час виконання дорівнює $T_k = \dfrac{\lambda_f^{\text{int}}}{kP\mu}$. Якщо $k > P$, то час обчислюється за формулою:

$$T_k = T_{\left[\frac{k}{P}\right]} + T_{k - \left[\frac{k}{P}\right]}, \qquad (3)$$

де $[a]$ – ціла частина числа a.

5. Експериментальне підтвердження результатів

Для перевірки отриманих результатів та обґрунтування подальших досліджень були проведені експерименти з обчислень добутку матриць на кластері Інституту програмних систем НАН України. Конфігурація експериментального середовища приведена в табл. 1.

Було взято матрицю з розмірами 4000 на 4000 елементів. Результати обчислень на одному процесорі для різних розрізань матриці приведено в табл. 2.

Таблиця 1

Операційна система:	Scientific Linux 5
Тип системи	64-бітна ОС
Процесори	2xIntel® Xeon® Processor E5405 Quad Core
Кількість ядер	4
Кількість потоків	4
Об'єм L2-кеша	12 Mb
Об'єм оперативної пам'яті	16 Gb

На основі даних табл. 2 виконано апроксимацію функції $f_c(\cdot)$ та інших параметрів моделі (1). В результаті ми отримали змогу оцінити час виконання задачі для довільної кількості однорідних процесорів. На рис. 4 показані криві – залежність часу виконання (вертикальна вісь) від способу розбиття (кількості блоків на які ділиться матриця – горизонтальна вісь) для різної кількості процесорів.

Таблиця 2

№	Кількість частин	Час на множення однієї частини (секунди)
1	1	2286.21
2	2	1072.28
3	4	683.25
4	8	260.25
5	16	130.43
6	20	103.41
7	25	82.97
8	32	63.69
9	40	51.03
10	64	32.48
11	80	26.04
12	100	20.81
13	128	14.8

Для перевірки моделі були здійснені експерименти обчислення часу виконання для 2 і 3 процесорів. На рис. 5 показані порівняння отриманих кривих.

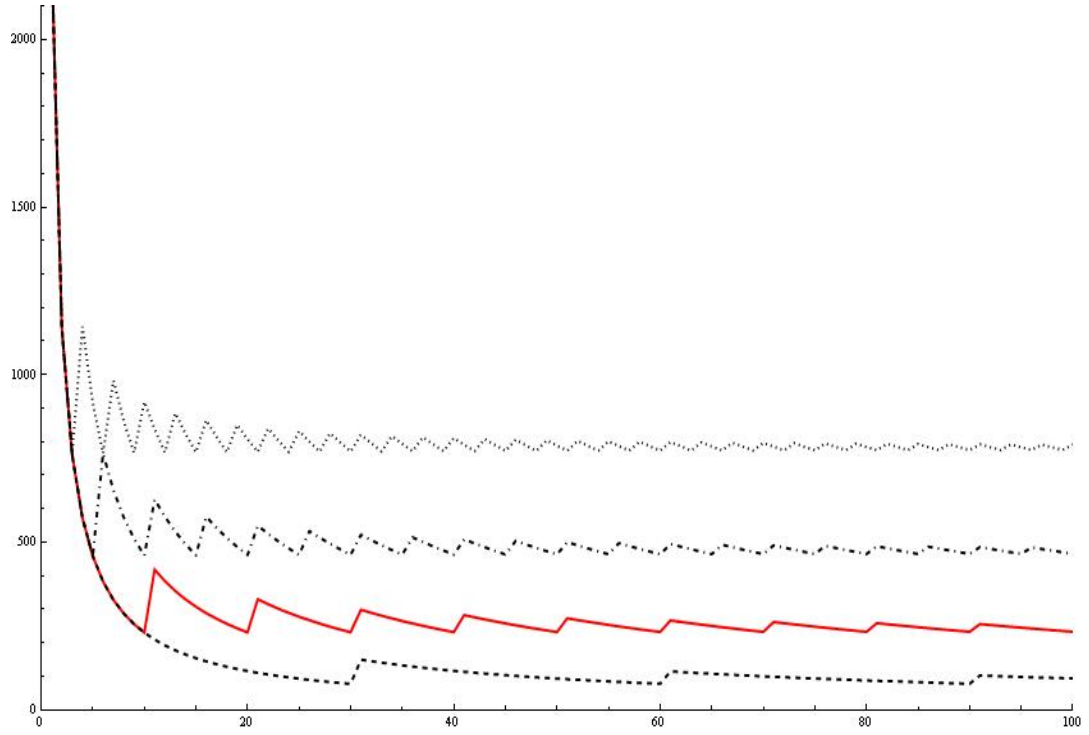

Рис. 4. Залежність часу виконання від способу розбиття для різної кількості процесорів

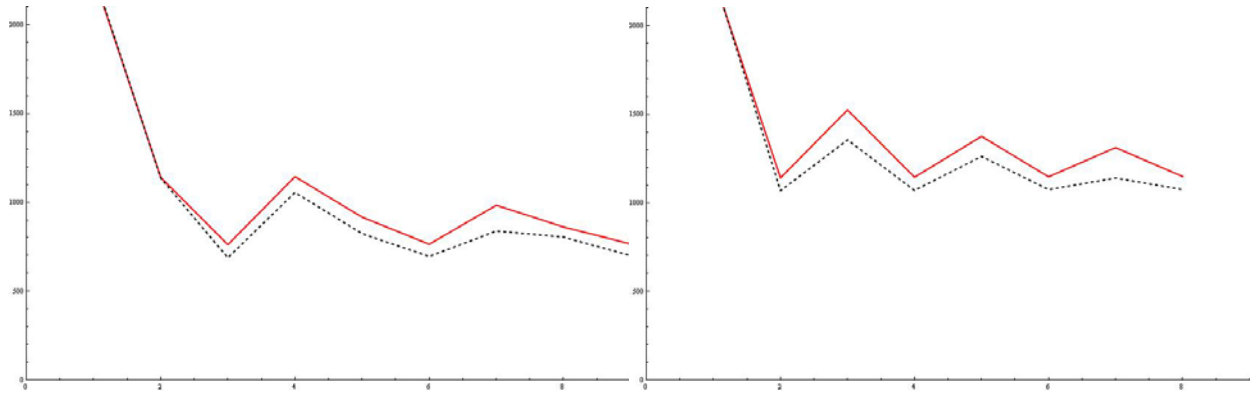

Рис. 5. Теоретична (неперервна лінія) і експериментальна (пунктирна лінія) криві для 3 і 2 процесорів

Висновки

У даній роботі досліджувалась модель керування ресурсами в однорідному багатопроцесорному середовищі. Запропоновано підхід на основі керованих потокових моделей, який дозволяє отримати оптимальне у сенсі швидкодії керування. Для задачі множення матриць було отримано аналітичний вид часу закінчення роботи в залежності від параметрів розпаралелювання та відповідне екстремальне керування. Виконані експерименти на кластері Інституту програмних систем підтверджують теоретичні результати роботи.

1. *Srinivasa Prasanna G.N., Musicus B.* Generalized Multiprocessor Scheduling Using Optimal Control // Proc. SPAA. – 1991. – P. 216 – 228.
2. *Foster I., Kesselman C.* The Grid: Blueprint for a new Computing Infrastructure. Morgan Kaufmann, 2004. – 676 p.
3. *Shivaratri N.G., Krueger P., Singhal M.* (1992, December). Load distribution for locally distributed systems. IEEE Computer 8 (12), P. 33 – 44.
4. *Chronopoulos A.T., Penmatsa S., Yu N.* Scalable Loop Self-Scheduling Schemes for Heterogeneous Clusters // Proc. of IEEE Intern. Conf. on Cluster Computing, Chicago, Illinois, USA, 2002. – P. 353 – 359.
5. *Tantawi A.N., Towsley D.* Optimal static load balancing in distributed computer systems // J. of the ACM, **32** (2), 1995. – P. 445 – 465.
6. *Tang X., Chanson S.T.* Optimizing static job scheduling in a network of heterogeneous computers // Proc. of the Intern. Conf. on Parallel Processing, 2002. – P. 373 – 382.
7. *Ross K.W., Yao D.D.* Optimal load balancing and scheduling in a distributed computer system // Journal of the ACM, **38** (3), 1991. – P. 676 – 690.
8. *Grosu D., Chronopoulos A.T.* A game-theoretic model and algorithm for load balancing in distributed systems // In Proc. of the 16th IEEE Intern. Parallel and Distributed Processing Symposium, Ft Lauderdale, Florida, USA, 2002. – P. 146 – 153.
9. *Grosu D., Chronopoulos A.T.* Noncooperative load balancing in distributed systems // J. of Parallel and Distributed Computing, **65** (9), 2005. – P. 1022 – 1034.
10. *Kwok Y.K., Hwang K., Song S.* Selfish grids: Game-theoretic modeling and nas/psa benchmark evaluation // IEEE Trans. on Parallel and Distributed Systems, **18** (5), 2007. – P. 621 – 636.
11. *Ghosh P., Roy N., Das S.K., Basu K.* A pricing strategy for job allocation in mobile grids using a non-cooperative bargaining theory framework // J. of Parallel and Distributed Computing, **65** (11), 2005. – P. 1366 – 1383.
12. *Kwok Y.K., Song S., Hwang K.* Selfish grid computing: Game-theoretic modelling and nas performance results // In Proc. of the CCGrid, 2005. – P. 1143 – 1150.
13. *Hui C.C., Chanson S.T.* (1999, July-Sept.). Improved strategies for dynamic load balancing. IEEE Concurrency, **7** (3), 1999. – P. 58 – 67.

14. *Campos L.M., Scherson I.* Rate of change load balancing in distributed and parallel systems // Parallel Computing, **26** (9), 2000. – P. 1213 – 1230.
15. *Corradi A., Leonardi L., Zambonelli F.* Diffusive load-balancing policies for dynamic applications // IEEE Concurrency, **7** (1), 1999. – P. 22 – 31.
16. Lee S.H., Hwang C.S. A dynamic load balancing approach using genetic algorithm in distributed systems // In Proc. of the IEEE Intl. Conf. on Evolutionary Computation, 1998. – P. 639 – 644.
17. *Nazarathy Y., Weiss G.* A Fluid Approach to Large Volume Job Shop Scheduling // J. of Scheduling, **13** (5), 2010. – P. 509 – 529.
18. *Meyn S.* Control Techniques for Complex Networks. – Cambridge University Press, 2007. – 582 p.
19. *Милютін А.А., Дмитрук А.В., Осмоловский М.П.* Принцип максимума в оптимальном управлении. – М.: Изд-во МГУ, 2004. – 168 с.

Отримано 28.01.2011

Про авторів:

Дорошенко Анатолій Юхимович,
доктор фізико-математичних наук,
професор,

Ігнатенко Олексій Петрович,
кандидат фізико-математичних наук,
старший науковий співробітник,
докторант,

Іваненко Павло Андрійович,
аспірант.

Місце роботи авторів:

Інститут програмних систем
НАН України,
03187, Київ - 187,
Проспект Академіка Глушкова, 40.
Тел.: 526 6025.
e-mail: o.ignatenko@gmail.com